COMMODORE MAGAZINE

VOL. 2
ISSUE 8

commodore
POWER
CLR HOME
INST DEL
RESTORE
RETURN
SHIFT
CRSR
CRSR

- ★ C64 Memory Maps
- ★ CBM Multi User
- ★ Vic Tiny Aid

Registered by Australia Post
Publication No. NBP4288

REGISTERED FOR POSTING AS A PUBLICATION: CATEGORY B

The objective of this magazine is to disseminate information to all users of Commodore computer products. This magazine contains a variety of information collected from other Commodore publications, and generated locally. Contributions from all Commodore User Groups, and individual users are encouraged.

Advertising space is restricted and allocated on a first-come, first-served basis. Advertising rates can be supplied on request.

All copy and advertising should be addressed to:

The Editor,
COMMODORE MAGAZINE,
P.O. BOX 336, ARTARMON
N.S.W. 2064
AUSTRALIA

ISSUE No.	COPY/ADV DEADLINE	PUBLICATION DATE
1	February 18th	March 6th
2	April 1st	April 17th
3	May 13th	May 29th
4	June 17th	July 3rd
5	July 29th	August 14th
6	September 9th	September 25th
7	October 14th	October 30th
8	November 25th	December 4th

Production & ypesetting:
Mervyn Beamish Graphics Pty. Ltd.
82 Alexander Street, Crows Nest
2065
Phone 439 1827

Printer:
Liberty Print
108 Chandos Street, Crows Nest
2067
Phone 43 4398

SUBSCRIPTIONS

	Annual Subscription	Single Copy
Postage paid within Australia	$A30.00	$A5.00
Overseas Postage Paid	$A38.00	$A6.00

Subscriptions to COMMODORE MAGAZINE can be obtained from your authorised Commodore Dealer, or from:

COMMODORE MAGAZINE,
P.O. BOX 336, ARTARMON,
N.S.W. 2064,
AUSTRALIA.

Vol 1 1981
Vol 2 1982

Typeset and assembled off Commodore Wordcraft disks

PLEASE NOTE: To provide a good information service to Commodore users, we will regularly mention equipment, software and services offered by companies and individuals not directly related to Commodore. In doing so, we are not making recommendations, and cannot be responsible for the validity and accuracy of any statements made.

EDITOR'S DESK . . .

The year has raced by and there have been many developments in Commodore world. The VIC-20 has become the leading home comuter in both USA and Australia with sales exceeding all expectations. This has resulted in an unprecidented growth in third party software for the Commodore range of microcomputers. There are many new programs being released for the VIC and the Commodore 64.

1983 plans will show many new developments as well. Early in the new year Commodore will announce a hand held computer that will put computing into everyone's pocket.

Also early in the new year will be the Australian release of the Commodore 64 followed by the release of a new series of business computers.

The Commodore Magazine will be undergoing a major change as well and the size, content and presentation will be upgraded to provide all the information you want and the subscription rate reduced to $A25.00 per annum if you use the form on page 25.

So to ensure continuity of your subscription, please return the subscription form on page 25 as soon as possible.

table of contents

Volume 2 Issue 8

Page	*Contents*
1	Introduction
2	The 12 Computerised days of Christmas
3	Everything you always wanted to know about Commodore Computers.
9	The Arbiter
14	Another Voice for the VIC
15	Tiny aid for VIC-20
20	Commodore 64K Memory expansion board
23	Accessing the SuperPET RS 232
25	Magazine Subscriptions
28	Canyon Bomber
30	Technical Notebook – DOS 1.2 Problems
32	Keyed Random Access
37	The VIC-20 Electonic Christmas Card
38	The Commodore 64: A Preliminary Review
52	Missing Page from Issue 7

REMEMBER TO RETURN YOUR COMPLETED SUBSCRITION FORM FOR VOLUME 3 (1983) — SPECIAL OFFER PAGE 25

list of advertisers

B.S. Microcomp	Inside Cover
Compute CBM Systems	Back Cover
CW Electronics	26 – 27
Mervyn Beamish Graphics	21
Micropro Designs	21
Pittwater Computer	29
The Microcomputer House	36

THE TWELVE COMPUTERISED DAYS OF CHRISTMAS

On the first day of Christmas, my computer gave to me
A glitch on the video screen.

On the second day of Christmas, my computer gave to me
Two keyboard bounces,
And a glitch on the video screen.

On the third day of Christmas, my computer gave to me
Three loose plugs,
Two keyboard bounces,
And a glitch on the video screen.

On the fourth day of Christmas, my computer gave to me
Four garbled saves,
Three loose plugs,
Two keyboard bounces,
And a glitch on the video screen.

On the fifth day of Christmas, my computer gave to me
Five blank cassettes,
Four garbled saves,
Three loose plugs,
Two keyboard bounces,
And a glitch on the video screen.

On the sixth day of Christmas, my computer gave to me
Six I/O spasms,
Five blank cassettes,
Four garbled saves,
Three loose plugs,
Two keyboard bounces,
And a glitch on the video screen.

On the seventh day of Christmas, my computer gave to me
Seven system re-sets,
Six I/O spasms,
Five blank cassettes,
Four garbled saves,
Three loose plugs,
Two keyboard bounces,
And a glitch on the video screen.

On the eighth day of Christmas, my computer gave to me
Eight worthless printouts,
Seven system re-sets,
Six I/O spasms,
Five blank cassettes,
Four garbled saves,
Three loose plugs,
Two keyboard bounces,
And a glitch on the video screen.

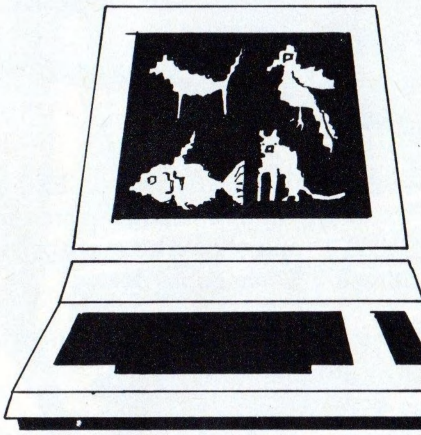
On the ninth day of Christmas, my computer gave to me
Nine burnt-out fuses,
Eight worthless printouts,
Seven system re-sets,
Six I/O spasms,
Five blank cassettes,
Four garbled saves,
Three loose plugs,
Two keyboard bounces,
And a glitch on the video screen.

On the tenth day of Christmas, my computer gave to me
Ten disc-drive lockouts,
Nine burnt-out fuses,
Eight worthless printouts,
Seven system re-sets,
Six I/O spasms,
Five blank cassettes,
Four garbled saves,
Three loose plugs,
Two keyboard bounces,
And a glitch on the video screen.

On the eleventh day of Christmas, my computer gave to me
Eleven damaged diskettes,
Ten disk-drive lockouts,
Nine burnt-out fuses,
Eight worthless printouts,
Seven system re-sets,
Six I/O spasms,
Five blank cassettes,
Four garbled saves,
Three loose plugs,
Two keyboard bounces,
And a glitch on the video screen.

On the twelfth day of Christmas, my computer gave to me
Twelve blown-out circuits,
Eleven damaged diskettes,
Ten disk-drive lockouts,
Nine burnt-out fuses,
Eight worthless printouts,
Seven system re-sets,
Six I/O spasms,
Five blank cassettes,
Four garbled saves,
Three loose plugs,
Two keyboard bounces,
And a glitch on the video screen.

EVERYTHING YOU ALWAYS WANTED TO KNOW ABOUT COMMODORE COMPUTERS *



* And Asked!

PET Questions

1. Q: How can the Pet 2001 be upgraded?

A: Commodore 2000 series machines can be upgraded by changing the BASIC ROM chips. The original 4K and 8K PET computers do not have ROMs to convert the operating system beyond BASIC 3.0. ROM chip sets (called "BASIC Upgrade ROMs") are available from most authorised Commodore dealers. Benefits to be gained by upgrading the machine are disk commands, enhanced BASIC interpreter, and a wider range of compatibility with available application software.

2. Q: Where can information be obtained about the PET 2001 parallel user port?

A: Information about the parallel user port of the PET 2001 can be found in The PET Revealed or in PET Interfacing (published by Howard W. Sams Co., Inc.). These books can be purchased from your authorised Commodore dealer.

3. Q: Can a 4016 be upgraded with additional RAM?

A: Commodore does not recommend or support memory upgrade modifications, although there are some vendors who make an expansion memory system available. Such a device connects though the memory expansion port.

4. Q: Is the character set now in the PET/CBM encoded in the memory chip?

A: The character set is contained in a chip called the character generator ROM (socket #UF10 in the 2001 and socket #A3 in the Fat Forty and the 8032).

5. Q: How can a TV set or a monitor be connected to a PET for use in a classroom?

A: The computers with a nine inch screen require a video adapter to generate a composite video signal and a monitor. Units with a twelve inch screen require the video adapter and a high resolution (16 MHz or greater) monitor.

6. Q: Can a repeating key be created with the graphics keyboard on the 4016 and the 4032?

A: A repeating key can be created on the 4016 and the 4032 if you write a machine language program for our version of BASIC. There is also a 4K ROM (called the Machine Language Utility-Pac) that will fit into an empty ROM slot of the machine which has an Auto-Repeat command included. More information about the Machine Language Utility-Pac is published in the Software Encyclopedia/Second Edition.

7. Q: We are using seven PETs, but have only two disk drives. Would it be possible to connect more than one PET to a disk drive?

A: Although it is not recommended, it is possible to connect more than one computer to a disk drive. Be careful to have only one computer access the disk drive at any one moment, or else the system may crash. One way to prevent this problem is to use the MUPET. This device allows connection of up to eight computers to one disk drive. Another alternative is the REGENT. This device allows connection of up to 16 computers to one drive. It works especially well in a school environment.

CBM Questions

1. Q: To which machines can the 64K Memory Expansion Board be connected?

A: The 64K add-on board can be used in conjunction with the 8032. It also works with the Fat Forty computer that is equipped with the universal logic board.

2. Q: What is the maximum usable RAM on the PET/CBM?

A: Both the PET and the CBM have 31,743 bytes of RAM available for use in programming.

3. Q: What is the MUPET and the REGENT?

commodore
CBM
Model 4032

A: The MUPET and the REGENT are devices that allow several computers to share a common disk drive. These devices switch the bus from computer to computer as the situation requires. Since they are hardware-only devices, they use no RAM in any of the computers.

4. Q: Can a typewriter be interfaced with an 8032?

A: Some typewriters can be interfaced with the 8032 with difficulty. For instance, a program must be written to translate characters to the kind required by the typewriter, and circuit boards may have to be changed. Interfaces used are RS232 or connections to the user port.

5. Q: What is the procedure to get into the graphics mode?

A: To get into the upper case/graphics mode on the 8032, simply PRINT CHR$(142). To get back into normal upper/lower case mode, PRINT CHR$(14).

6. Q: What clock rate is used on the 8032? Why isn't a 4 MHz used?

A: The 8032 uses a 1 MHz clock. The internal architecture of the 6502 chip is more efficient than that of other microprocessors in that the instruction fetch/decode cycle overlaps the execution cycle. Thus a 1 MHz 6502 is faster than a 2 MHz 8080 or Z80 microprocessor.

7. Q: Where is the cursor address stored in the 8032?

A: On the 8032, the column position of the cursor is decimal address 198. The row position is decimal address 216.

8. Q: How can the machine language monitor be entered?

A: To enter the machine language monitor, use the SYS command to jump to any location containing a binary zero. For example: SYS 4 or SYS 1024. This works on any PET/CBM computer that contains the built-in machine language monitor. This includes all but the first machines – and those machines that have a monitor on tape.

9. Q: How does Commodore's IEEE differ from Hewlett Packard's IEEE?

A: There are several differences in both hardware and software. Commodore features a Remote Enable line that is tied permanently to ground. The Interface Clear line is not used and the Commodore Service request line is not implemented in the firmware. Also, the HP does not include a 64 millisecond time-out on their bus. For more information, refer to The PET and the IEEE-4888, which is published by Osborne/McGraw Hill, and should be available at your local Commodore dealer.

10. Q: How can a window be set on the 8032?

A: Under program control, move the cursor to the upper-left corner of your desired window and PRINT CHR$(15). Then move the cursor to the lower-right corner of the window and PRINT CHR$(143). This procedure is elaborated in the PET/CBM Personal Computer Guide/Second edition.

11. Q: What is the difference between ULSL ASCII code and Commodore PET ASCII?

A: U.S. ASCII code uses a seven bit code which can represent 128 characters (2=128). PET ACSII has an eight bit code which can represent 256 characters (2=256); this accounts for the additional characters on Commodore computers. This is why an ASCII converter is necessary to interface with third party printers.

12. Q: Can a CBM computer run a VIC 20 Program?

A: Our CBM range of computers will run a 3K expanded VIC 20 program with no difficulties. To run a program from an unexpanded VIC 20 the following steps are necessary:-

POKE41,16 (moves BASIC pointer to start of BASIC on CBM)
POKE4096,0 (initialises the new BASIC area with zero).

To run an 8K or 16K program the following steps are necessary:-

POKE41,18
POKE4607,0
CLR

Your VIC 20 programs will now run on the CBM computer!

13. Q: Are spaces necessary in the BASIC text of a line?

A: The Commodore computer will disregard unnecessary spaces in a program. Spaces are mainly used for clarity and easy reading of programs.

14. Q: How can one program be loaded from another program?

A: The first program must incorporate a load statement in the last executed statement of the program.

---

commodore
SuperPET
SP9000
computer

SUPERPET Questions

1. Q: What is resident in the SuperpET that allows for multiple high-level languages?

A: The ROMs in the SuperPET contain routines that handle system functions such as input/output, screen handling, keyboard input, and general utility functions. These routines interface with the operating system for the particular language that you load into the computer. This allows different languages to be loaded which use the same utility routines.

2. Q: Will the languages available for the SuperPET work with a 4040 disk drive?

A: The languages will work, but the diskettes supplied are in the 8050 format, so they would have to be copied down to 4040 format first.

3. Q: Can an 8032 be upgraded to be a SuperPET?

A: In the future, a "single-board" upgrade will be available from your Commodore dealer. This will include language disks and the manuals.

4. Q: Why won't VISICALC and other software that uses ROM chips run on the SuperPET?

A: The add-on memory occupies the same address space as ROM slot #UD12. The 6502 "sees" RAM there instead of the ROM chip. A dealer-installed upgrade is available.

5. Q: When using the modem with the SuperPET, why do characters seem to be echoed back onto the screen?

A: The built-in communications program on the SuperPET prints the characters on the screen as they are keyed in from the keyboard. If the host computer is set to echo, then the received character will be a duplicate, causing the screen to display each character twice. Almost every host computer can be stopped from echoing. The specific command depends on the host computer being used.

5. Q: Are the SuperPET languages (FORTRAN, APL, BASIC, PASCAL, and Assembler) limited subsets of the languages?

A: All of the languages included with the SuperPET are complete languages written by the University of Waterloo.

6. Q: Is debugging available for the languages included with the SuperPET?

A: Debugging is included with all of the SuperPET languages. Debugging allows the computer to work interactively with the programmer in editing and correcting the program.

7. Q: How much memory is used when the languages are inserted into internal memory of the SuperPET?

A: The high-level languages load the interpreter into the upper 64K of RAM and leave the lower 32K of RAM for programming.

8. Q: Are the SuperPET languages stored as interpreters or as compilers?

A: All of the Waterloo languages at this time are interpretive languages which means that the computer "compiles" and executes the program on a line-by-line basis. This saves time in the debugging process.

9. Q: How can the screen be cleared from within the program on the SuperPET?

A: Printing the chr$(12) character in 'Waterloo MicroBASIC' will clear the screen and home the cursor.

10. Q: What pins are required to connect a modem to the SuperPET?

A: The SuperPET has pins 1,2,3,4,5,6, 7,8 and 20 for use. The SuperPET can be used to interface with serial devices other than modems. However, depending upon which particular device is being used, as few as three (1,2 and 3) or a combination up to nine pins may be needed. The pin configuration will be dictated by the peripheral being connected to the SuperPET. For connecting to a modem, a direct connect 25 pin cable is recommended.

11. Q: How do you access the COBOL interpreter on the new Update disk?

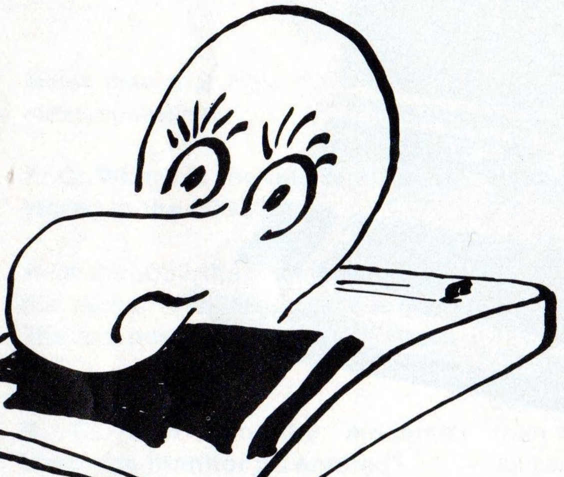
A: The menu displayed on the screen does not show a selection for COBOL. Replacement ROMs to change the screen menu will not be available through Commodore. The Cobol interpreter is loaded by typing, using the shift key, COBOL.

12. Q: Is there a networking system for the SuperPET?

A: There is, at this time, no true networking system for the SuperPET. Until one beomes available, the MUPET and DOUBLE MUPET from CMD should satisfy most users.

13. Q: What baud rates can be used with the SuperPET?

A: The SuperPET is capable of transmitting from 50 to 9600 baud. Refer to the SuperPET overview Manual, pages 60 and 61 for 'Setup' options.



VIC 20 Personal Computer Questions

1. Q: Can software programs made by other vendors (Atari/Radio Shack, etc.) be utilised by the VIC 20?

A: In general, software that has been written for a particular computer can be used only on that computer. Therefore, software programs made by other vendors cannot be used on the VIC 20.

2. Q: How can programs saved on the 8032 be loaded to the VIC and vice versa?

A: The VIC will load PET/CBM programs from tape without any modification if there is enough memory installed in the VIC. To load a PET/CBM with a VIC program, first POKE41, 16, then POKE 4096,0, then type NEW and proceed to load.

3. Q: Is it possible to use multiple joysticks with the VIC?

A: Only one joystick can be plugged into the game control port. Any additional joysticks must be interfaced to the user port with the appropriate software.

4. Q: Can more than one disk drive be connected to the VIC while using the printer?

A: Up to five disk drive units can be "daisy-chained" together. To include a VIC Printer in the system, simply connect it as the last unit of the chain.

5. Q: How is the RUN/STOP key disabled on the VIC 20?

A: Disable-POKE788,194. Enable-POKE788,191.

6. Q: What does EMAIL stand for when accessing a telecommunications network?

A: EMAIL stands for Electronic Mail. This is a means of sending and receiving messages through telecommunication lines.

DISK DRIVES Questions

1. Q: What do the lights on the disk drive signify?

A: The light on the top centre unit indicates an error condition if it turns red and stays on. (Flickering is normal during operation.) The lights on each drive indicate activity in that particular drive.

2. Q: There is a buzzing noise during initialisation of a disk – is something wrong with the disk drive?

A: The noise is normal. The head mechanism is creating the noise by vibrating the head against the travel limit stops. This ensures that track number one has been located.

3. Q: What kind of diskettes should be used with the disk drive?

A: We suggest using double-density diskettes with the 8050 and the 8250 because they tend to be slightly more reliable than others. However, any good-quality, soft-sectored, single-density diskette will work well with any model disk drive.

4. Q: How can the disk drive be cleaned?

A: The disk drive can be cleaned by using a commercial head-cleaning diskette and cleaning fluid. These products are available from most computer or office equipment stores.

5. Q: What is the cause of READ and LOAD errors?

A: A number of things can cause errors. Possible causes include:
1) improperly seated diskette;
2) physical or electrical damage to the diskette;
3) dirty heads in disk drive; and
4) drive head misaligned or speed of rotation not properly set. Your Commodore dealer is equipped to evaluate these problems if they persist.

6. Q: How can the 2040 disk drive be upgraded to a 4040 disk drive?

commodore
CBM
Model 8050
dual drive
floppy disk
drive 1
drive 0

A: A 2040 can be upgraded by simply replacing the ROM chips on the main logic board. Consult your Commodore dealer for ordering information.

7. Q: Can the 8050 disk drive be upgraded to an 8250 disk drive?

A: The 8050 cannot be upgraded to an 8250, however, the diskettes are compatible. Of course, the 8050 only uses (reads/writes) the bottom side of the diskette. The 8250 uses both sides. Therefore, it is the bottom side of the diskette that is compatible with both drives.

8. Q: Can a 4040 dual disk drive read a diskette made on a 2040 disk drive?

A: A 2040 diskette can be read from a 4040 disk drive, but it cannot be written on. The means that a 4040 drive is read compatible but not write compatible with a 2040 diskette.

9. Q: How can a 4040 diskette be copied to an 8050 disk format for use on the 8050 drive?

A: The following procedure lists the proper steps:
1) Turn on the computer and the 8050 drive only;
2) Run the "Change 8050" program (included on the Test/Demo diskette);
3) Turn on the 4040 drive;
4) Run the "Unit to Unit" program (found on the Test/Demo diskette). The Unit to Unit program will copy all of your files from the 4040 diskette to the 8050 diskette.

10. Q: How many files will each diskette hold?

A: A 4040 diskette is capable of saving a maximum of 144 files. An 8050 diskette is capable of saving a maximum of 224 files. These files may be either sequential, program, relative, or a combination. Once that limit has been reached, the directory is full – even if the rest of the disk is not.

11. Q: Will the 8250 Disk Drive accept 8050 Disk Drive diskettes and vice versa?

A: The 8050 diskettes are read/write compatible when used on the 8250 DisK Drive. When using and 8250 diskette in the 8050 Disk Drive only the underside of the 8250 diskette will be read/write compatible.

12. Q: What is the BAM?

A: A BAM stands for Block Availability Map. This is a disk memory representation of available and allocated space on disk. It is referenced by the DOS (Disk Operating System) to determine what space is available and how many blocks can be allocated.

13. Q: In disk operating commands, why does a "d" precede certain commands?

A: Machines using BASIC 4.0 need this preceding the command.

14. Q: What is a Hard Disk?

A: A hard disk is one that has a rigid platter on which the magnetic media iron oxide is coated.

PRINTERS Questions

1. Q: What does the blinking light signify on the 4022?

A: When the red light blinks, it indicates that the printer is out of paper or that the paper is not inserted correctly.

2. Q: Are there bi-directional printers available from Commodore?

A: There are printers available with the bi-directional feature such as the 8023P and the 8024. The 4022 printer can be upgraded by replacing ROM with an upgrade ROM. The part number of this upgrade ROM is 901631-02.

3. Q: Is the 4022 upward compatible with the 8023?

A: Programs that work with a 4022 will also work with the 8023.

4. Q: Will an 8032 accept two printers?

A: The 8032 can be used with up to four printers connected.

5. Q: Is the 8300 printer supplied with a tractor feed mechansim?

A: The tractor feed mechanism is an optional feature for the 8300 printer, and it allows bi-directional (up/down) movement of the paper. This feature is available from your Commodore dealer.

6. Q: What cables are needed to connect a Commodore printer to a Commodore computer?

A: In order for the first peripheral to be connected to a Commodore computer, a PET to IEEE cable is required. For every additional peripheral, an IEEE to IEEE cable is required. For this reason, the cables are sold separately from the peripherals.

7. Q: Do any Commodore printers have graphics capability?

A: The Commodore 1515, 2022, 2023, 4022, and 8023 printers all support PET graphic characters.

commodore
Tractor Printer
8023P

8. Q: Do any Commodore printers print characters with true ascenders and descenders?

A: With the exception of the VIC 1515 printer, all Commodore printers are capable of printing characters with true ascenders and descenders.

9. Q: What type of printer heads do the 2022 and 4022 printers use, and what is their life expectancy?

A: The 2022 printer uses a print head that can be repaired. The 4022 printer uses a print head that can be replaced once it is worn out. The life expectancy of each is approximately 60 million characters.

10. Q: How can I get a listing of my program on the printer?

A: To get a listing of a program do the following: depress the 'shift key' and '5 key' to put the language editor in command mode, then for;

(1) BASIC – save 'printer'.
(2) COBOL -p'printer'.
(3) FORTRAN – p'printer'.
(4) PASCAL – p'printer'.

11. Q: How can I use the printer within my program to list data?

A: Each language interpreter accesses the printer in a different way. Rather than give programs listings to show how the printer is accessed, I will indicate modifications that can be made to the 'tutorials' to use the printer.

(1) BASIC – Page 53, Example-29. Change line 40, '"namefile"' to '"printer"'. To get a clean run when executing this program DELETE lines 140-220. The printer will list names entered via the console.

(2) COBOL – Page 64-68, Example-18. This program shows exactly how the printer is used to "Print a report on the printer".

(3) FORTRAN – Page 61, Example-39. Change line 5 (be sure to count the blank line) ""namefile"" to '"printer"'. Again for a clean run eliminate the lines of code from the second 'open...' statement through the 'close...' statement.

(4) PASCAL – Page 7, Example-5. This program requires some additional defining in the VAR section. Between the "var" and "begin" lines enter "p:text;". After the first "begin" line, enter "rewrite (p,'rinter');". You might want to refer to page 93 for additional information regarding "var" and "rewrite" usage within a program. The "p" becomes the first parameter within parenthese of the "writeln" statements: "writeln(p,'A table of squares and cubes:');" and "writeln(p,x, xsquared, xcubed);".

12. Q: How can a program be listed in upper/lower case on the 8023 printer?

A: Set the printer to upper/lower case by using this sequence of commands:-

```
OPEN 1,4,7
PRINT #1
CLOSE1
```

The printer now prints in upper/lower case.

THE ARBITER

Originated by Gippsland Computer Business Services
167 Princes Highway,
BAIRNSDALE.
ph (051) 525939

With a couple of EPROMs, a few bits of wire and edge sockets, it is relatively easy to produce multi terminal Commodore computers that share the IEEE bus. This method is simple, requires minimal software changes, and most importantly, it is ***super cheap****. We have used this with the CBM9090 hard disk drive with raging success. It is so quick at opening and closing files, that programs that do not hog the bus work incredibly well. The ARBITER system is intended for applications of up to 3 or 4 terminals maximum but can arbitrate the bus for up to 255.*

THE BAD NEWS OUT OF THE WAY

Like all cheap tricks, there are a few drawbacks, so for a change here are the known limitations and outright bugs discussed first. Once we get that out of the way we can brag with a clear conscience.

Only one computer can use the bus at a time. It is important to use the bus and then get off it as quickly as you can. You can handle this in your programs by shutting files as soon as possible.

If someone presses the stop key just after a SHIFT/RUN has been pressed you will have to quickly enter open15,8,15:close15. However if another user grabs the bus in between whiles... then resetting of all CPUs is needed. This is a nasty one. Any ideas anyone???

If used with an SP9000 the presence of an EPROM in the F-ROM position gave us a problem. Is seems that with all the other 6809 ROMs lying across the address lines, the capacitance increased too much and we couldn't get it to bank select the 6809 software properly. So we soldered the new F-ROM on top of the old one and put in a switch for the select line on pin 20.

Machine code programs that have their own bus handling routines or use the bus in strange ways will not work. The two we know of are WORDCRAFT and SILICON OFFICE. WORDCRAFT has a new version that already allows multi terminal and old versions are very easily changed if Peter Dowson and Commodore don't mind. SILICON OFFICE is the same story, but we don't know how to restore the diskette protection once it has been modified.

HOW TO INSTALL THE GADGET

1. String the cable between the userports of the computers that will be sharing. Daisy chain or star configuration is acceptable. Try the loop system (the vatican ok's this one) but we haven't given that one a go. You only need 8 wire ribbon cable that connects the 8 data lines. These are underneath the circuit board and looking from the rear of the unit, start at the third and go left to right.
2. Make two EPROMs for each computer you will be using. The listing provided gives all the info on what is needed for this. If it is a drag to make the bits and pieces, give us a ring and we can get them to you for cost plus and agro charge and post.

HOW IT BEHAVES IN USE

Turn on all the computers and devices on the bus. Whoever does a shift/run or load first gets the bus until the program is loaded. While the program loads the others then sit on a cue waiting. Each CPU has a weighting factor in the F-ROM chip. This means that if two are waiting for the bus at once then, the one with the smallest weighting is most likely to get the bus first. While you are waiting for the bus, a tick will be displayed on the right hand top of your screen. Once you get the bus, whatever character was there before the tick will be replaced. The longest you have to wait in normal use will be the time it takes for another computer to load a module. If the program on another computer opens a file and then leaves it open for a long period, it is considered to be hogging the bus. We recommend that any programs that do this be changed so they close files as soon as possible. This is good programming practice anyway, as files opened for writing usually leave unwritten data in the disk memory and if the power fails, you lose this data.

With the CBM9090 hard disk drive, waiting is very short. The application we are using it for involves medical billing. Two terminals can be billing concurrently. The bus is hogged the worse at the time the account statement is printed and is only a few seconds anyway. With a decent printer that has a buffer, this problem is gone too. In fact, the user spends so much time doing the entries for the account that the time spent waiting for the bus is negligible by comparison.

If you are writing programs and so on, and you leave a file open and drop to ready mode, the ARBITER will ring the bell once and print the message 'hog mode' in reverse screen in the line above the ready. Unless vital, we recommend that you close your open file(s) and allow the other users access to the bus. Any disk commands from direct mode are catered for by the ARBITER as well.

With the release of hard disk drives there will probably be a need for multi terminal access and the ARBITER is public domain so help yourself. The ony thing we ask, is that if you discover any bugs or clever enhancements, we would appreciate knowing. This is the only exchange we ask.

```
0:ARBITER.SRC......PAGE 0001

LINE# LOC   CODE        LINE

0001  0000              ;
0002  0000              ;=====================================
0003  0000              ;             THE ARBITER
0004  0000              ;      JOHN GYFFYN AND STEPHEN LAMB
0005  0000              ; GCBS P.O. BOX 200 BAIRNSDALE 3075 VICTORIA
0006  0000              ;          PHONE (052) 52 5939
0007  0000              ;=====================================
0008  0000              ;
0009  0000              ; 1. GRAB THE BUS WHEN OPENING AN IEEE FILE BY
0010  0000              ;    CHANGING F-ROM AT F595. REPLACE JMP F4A5
0011  0000              ;    WITH A JMP FD60
0012  0000              ;
0013  0000                   *=$FD60          ;START ADDRESS OF EXTRA STUFF
0014  FD60  08          OPEN PHP              ;CALLED FROM F595 - FILE OPEN
0015  FD61  48               PHA              ;PARANOID SAVING OF REGISTERS
0016  FD62  8A               TXA
0017  FD63  48               PHA
0018  FD64  98               TYA
0019  FD65  48               PHA
0020  FD66  20 FD FD         JSR GETBUS       ;HANG AROUND TILL BUS GRABBED
0021  FD69  68               PLA              ;COVER TRACKS BY RESTORING REGISTERS
0022  FD6A  A8               TAY
0023  FD6B  68               PLA
0024  FD6C  AA               TAX
0025  FD6D  68               PLA
0026  FD6E  28               PLP
0027  FD6F  A5 D3            LDA $D3          ;COPY MICROSOFT EXITS AT F4A5
0028  FD71  30 03            BMI EXIT
0029  FD73  4C A9 F4         JMP $F4A9
0030  FD76  60          EXIT RTS
0031  FD77  EA               NOP              ;LEAVE A BIT OF SPACE IF LATER
0032  FD78  EA               NOP              ;FIXES OR ENHANCEMENTS NEEDED
0033  FD79  EA               NOP
0034  FD7A              ; ================================
0035  FD7A              ;2. THIS ONE FIXES THE BUS WHEN JUST AFTER
0036  FD7A              ;   MICROSOFT HAS CLOSED A FILE.  THE BUS
0037  FD7A              ;   IS GIVEN UP ONLY IF ALL FILES TO IEEE
0038  FD7A              ;   ARE CLOSED.  THE F-ROM GETS CHANGED AT
0039  FD7A              ;   F32E AND F331 BY PUTTING JMP FD7A
0040  FD7A              ;   AND JMP FD80 THERE RESPECTIVELY
0041  FD7A              ;
0042  FD7A  B9 65 02    CLOSE LDA $0265,Y     ;CALLED FROM F32E
0043  FD7D  9D 65 02         STA $0265,X      ;PUT BACK WHAT WE STOLE
0044  FD80  08               PHP              ;CALLED FROM F331
0045  FD81  48               PHA              ;YES THE REGISTERS AGAIN
0046  FD82  8A               TXA
0047  FD83  48               PHA
0048  FD84  98               TYA
0049  FD85  48               PHA
0050  FD86  20 2B FE         JSR OFFBUS       ;GET OFF THE BUS IF YOU CAN
0051  FD89  68               PLA              :RUB OUT ANY SIGN OF YOUR PRESENCE
0052  FD8A  A8               TAY
0053  FD8B  68               PLA
0054  FD8C  AA               TAX
0055  FD8D  68               PLA
```

```
0:ARBITER.SRC......PAGE 0002

LINE# LOC   CODE        LINE

0056  FD8E  28                  PLP
0057  FD8F  60                  RTS
0058  FD90  EA                  NOP                  ;LEAVE ROOM IF NEEDED LATER
0059  FD91  EA                  NOP
0060  FD92  EA                  NOP
0061  FD93              ; ==========================================
0062  FD93              ;3. CHECK ON EVERY BASIC STATEMENT EXECUTED
0063  FD93              ;   AND PINCH THE BUS IF IT IS AND IEEE KEYWORD.
0064  FD93              ;   THIS ONE IS IN THE B-ROM AND YOU NEED TO
0065  FD93              ;   CHANGE THE BYTES AT B79F TO JMP FD93
0066  FD93              ;
0067  FD93  08          EXEC    PHP                  ;CALLED FROM B79F
0068  FD94  48                  PHA                  ;THE REGISTERS AGAIN
0069  FD95  8A                  TXA
0070  FD96  48                  PHA
0071  FD97  98                  TYA
0072  FD98  48                  PHA
0073  FD99  C0 48               CPY #$48             ;IS THE KEYWORD A 4.0 DOS ONE?
0074  FD9B  B0 05               BCS NOTEST
0075  FD9D  20 2B FE    TEST    JSR OFFBUS           ;NOT IEEE KEYWORD
0076  FDA0  F0 03               BEQ KEEP2            ;FILES OPEN SO HOG THE BUS
0077  FDA2  20 FD FD    NOTEST  JSR GETBUS           ;IEEE KEYWORD SO GRAB BUS
0078  FDA5  68          KEEP2   PLA                  ;PUT BACK THE PIECES
0079  FDA6  A8                  TAY
0080  FDA7  68                  PLA
0081  FDA8  AA                  TAX
0082  FDA9  68                  PLA
0083  FDAA  28                  PLP
0084  FDAB  4C 70 00            JMP $0070            ;CONTINUE WITH MICROSOFT
0085  FDAE  EA                  NOP                  ;THE USUAL 'FIX IT' SPACE
0086  FDAF  EA                  NOP
0087  FDB0  EA                  NOP
0088  FDB1              ; ==========================================
0089  FDB1              ;4. HANDLE THE DROPPING OUT TO READY STATE
0090  FDB1              ;   BY GIVING UP THE BUS IF YOU CAN.  IF ANY
0091  FDB1              ;   FILES ARE OPEN THEN CALL THE USER A BUS
0092  FDB1              ;   HOGGER SO HE CAN LET HIS MATES HAVE A GO
0093  FDB1              ;   BY CLOSING HIS FILES IF HE CAN.
0094  FDB1              ;   CHANGE THE BYTES AT B403 TO JMP FDB1
0095  FDB1              ;
0096  FDB1  08          READY   PHP                  ;CALLED FROM B403
0097  FDB2  48                  PHA                  ;THE USUAL REGISTERS SAVE
0098  FDB3  8A                  TXA
0099  FDB4  48                  PHA
0100  FDB5  98                  TYA
0101  FDB6  48                  PHA
0102  FDB7  AE 43 E8            LDX 59459            ;GOT THE BUS?
0103  FDBA  E8                  INX
0104  FDBB  D0 13               BNE FIN
0105  FDBD  20 2B FE            JSR OFFBUS           ;YEAH SO DROP IT IF CAN
0106  FDC0  AD 43 E8            LDA 59459
0107  FDC3  F0 0B               BEQ FIN              ;GOT DROPPED SO NORMAL READY
0108  FDC5  A2 0C               LDX #$0C             ;PRINT HOG MODE BEFORE READY PRINTED
0109  FDC7  BD D8 FD    PRINT   LDA HOG-1,X
0110  FDCA  20 D2 FF            JSR $FFD2
```

```
0:ARBITER.SRC......PAGE 0003

LINE# LOC   CODE        LINE


0111  FDCD  CA                 DEX
0112  FDCE  D0 F7              BNE PRINT
0113  FDD0  68          FIN    PLA              ;LEAVE ALL AS YOU FOUND IT
0114  FDD1  A8                 TAY
0115  FDD2  68                 PLA
0116  FDD3  AA                 TAX
0117  FDD4  68                 PLA
0118  FDD5  28                 PLP
0119  FDD6  4C 1D BB           JMP $BB1D        ;BACK TO BASIC
0120  FDD9  92          HOG    .BYTE $92, 'EDOM GOH', $12, $07, $0D
0120  FDDA  45 44
0120  FDE2  12
0120  FDE3  07
0120  FDE4  0D
0121  FDE5  EA                 NOP
0122  FDE6  EA                 NOP
0123  FDE7  EA                 NOP
0124  FDE8              ; ==========================================
0125  FDE8              ;5. THIS ONE FIXES THE LOAD, SAVE AND VERIFY
0126  FDE8              ;   STATEMENTS DONE FROM PROGRAM OR DIRECT
0127  FDE8              ;   MODE.  CHANGE THE BYTES AT F4AD TO
0128  FDE8              ;   JMP FDE8
0129  FDE8              ;
0130  FDE8  08          PROG   PHP              ;CALLED FROM F4AD
0131  FDE9  48                 PHA
0132  FDEA  8A                 TXA
0133  FDEB  48                 PHA
0134  FDEC  98                 TYA
0135  FDED  48                 PHA
0136  FDEE  20 FD FD           JSR GETBUS       ;KNOCK OFF THE BUS
0137  FDF1  68                 PLA
0138  FDF2  A8                 TAY
0139  FDF3  68                 PLA
0140  FDF4  AA                 TAX
0141  FDF5  68                 PLA
0142  FDF6  28                 PLP
0143  FDF7  4C D5 F0           JMP $F0D5        ;ALWAYS FINISH WHAT YOU STARTED
0144  FDFA  EA                 NOP
0145  FDFB  EA                 NOP
0146  FDFC  EA                 NOP
0147  FDFD              ; ==========================================
0148  FDFD              ;6. SUBROUTINE TO GRAB THE BUS.  TRY AND GET
0149  FDFD              ;   IT AND IF YOU CAN'T PUT A TICK AT THE TOP
0150  FDFD              ;   RIGHT CORNER OF THE SCREEN TO TELL THE
0151  FDFD              ;   USER THE COMPUTER IS STILL AWAKE DESPITE
0152  FDFD              ;   ALL APPEARANCES.
0153  FDFD              ;
0154  FDFD  AE 43 E8    GETBUS LDX 59459        ;CALLED FROM STUFF ABOVE
0155  FE00  E8                 INX
0156  FE01  F0 24              BEQ GOTIT        ;BEAUDIE! ALREADY GOT THE BUS
0157  FE03  AC 4F 80           LDY $804F        ;SAVE THE SCREEN CHARACTER
0158  FE06  A2 7A              LDX #$7A
0159  FE08  8E 4F 80           STX $804F        ;PUT A CUTE TICK THERE
0160  FE0B  A9 00       TRY    LDA #$00
0161  FE0D  8D 43 E8           STA 59459        ;SET GOT THE BUS BYTE
0162  FE10  AE 40 FE           LDX $FE40        ;GET DELAY AMOUNT
0163  FE13  CA          WAIT   DEX
0164  FE14  D0 FD              BNE WAIT         ;FREE FOR GIVEN TIME
0165  FE16  CE 43 E8           DEC 59459        ;OUTPUT MODE
0166  FE19  AD 41 FE           LDA $FE41        ;CPU NUMBER
0167  FE1C  8D 4F E8           STA 59471        ;PUT CPU NUMBER ON PARRALLEL PORT
0168  FE1F  CD 4F E8           CMP 59471        ;DID IT GET THERE OK?
0169  FE22  D0 E7              BNE TRY          ;NO SO TRY....TRY AGAIN
0170  FE24  8C 4F 80           STY $804F        ;GOT 'IM PUT BACK THE SCREEN BYTE
```

```
0:ARBITER.SRC......PAGE 0004

LINE# LOC   CODE          LINE

0171  FE27  60            GOTIT  RTS
0172  FE28  EA                   NOP
0173  FE29  EA                   NOP
0174  FE2A  EA                   NOP
0175  FE2B                ;==========================================
0176  FE2B                ;7. SUBROUTUINE TO DROP THE BUS IF THERE ARE
0177  FE2B                ;   NO OTHER FILES OPEN.  OTHERWISE HOG IT
0178  FE2B                ;   TO PREVENT THE OTHER CPU DROPPING BYTIES
0179  FE2B                ;   ALL OVER YOUR LOVELY FILES IN A MOST
0180  FE2B                ;   UNDIGNIFIED FASHION
0181  FE2B                ;
0182  FE2B  A6 AE         OFFBUS LDX $AE
0183  FE2D  F0 0A                BEQ GIVEUP
0184  FE2F  BD 5A 02      LOOK   LDA $025A,X        ;LOOK AROUND THE FILE TABLE
0185  FE32  C9 04                CMP #$04           ;GOT AN IEEE OPEN AT THE MOMENT?
0186  FE34  B0 06                BCS KEEP           ;YES SO HANG ON LIKE GRIM DEATH
0187  FE36  CA                   DEX
0188  FE37  D0 F6                BNE LOOK
0189  FE39  8E 43 E8      GIVEUP STX 59459          ;CHUCK THE BUS AWAY
0190  FE3C  60            KEEP   RTS
0191  FE3D  EA                   NOP
0192  FE3E  EA                   NOP
0193  FE3F  EA                   NOP
0194  FE40                ;
0195  FE40                       *=$FE40
0196  FE40  10                   .BYTE $10          ;WEIGHTING ;PUT THESE BYTES HERE
0197  FE41  02                   .BYTE $02          ;CPU NUMBER ;THESE ARE DECIDED AT
                                                     INSTALLATION
0198  FE42                ;==========================================
0199  FE42                ; 8. THIS ONE WAS FORGOTTEN THE FIRST TRY WE MADE
0200  FE42                ;    AT THIS CHEAP TRICK.  THE DS AND DS$ GET THE
0201  FE42                ;    BUS TOO.  CHANGE BYTES AT FFBD TO JMP FE50
0202  FE42                ;
0203  FE42                       *=$FE50
0204  FE50  08            DS     PHP                ;CALLED FROM FFBD
0205  FE51  48                   PHA
0206  FE52  8A                   TXA
0207  FE53  48                   PHA
0208  FE54  98                   TYA
0209  FE55  48                   PHA
0210  FE56  20 FD FD             JSR GETBUS         ;GRAB IT
0211  FE59  68                   PLA
0212  FE5A  A8                   TAY
0213  FE5B  68                   PLA
0214  FE5C  AA                   TAX
0215  FE5D  68                   PLA
0216  FE5E  28                   PLP
0217  FE5F  4C 95 D9             JMP $D995          ;AND RUN
0218  FE62                       .END

ERRORS = 0000


SYMBOL TABLE

SYMBOL VALUE
 CLOSE    FD7A    DS       FE50   EXEC     FD93   EXIT     FD76
 FIN      FDD0    GETBUS   FDFD   GIVEUP   FE39   GOTIT    FE27
 HOG      FDD9    KEEP     FE3C   KEEP2    FDA5   LOOK     FE2F
 NOTEST   FDA2    OFFBUS   FE2B   OPEN     FD60   PRINT    FDC7
 PROG     FDE8    READY    FDB1   TEST     FD9D   TRY      FE0B
 WAIT     FE13
END OF ASSEMBLY
```

Another Voice for the VIC

Normally, your VIC has 4 musical voices . . . three music registers and a white noise register . . . but by connecting a small amplifier and speaker to the USER PORT, and doing a little programming, you can get *another* musical voice.

The user port on the VIC is very similar to the user port on the PET. This makes it easy to adapt some of the PET's music methods to the VIC.

Background—Adding Sound to Older PET/CBM's
Before Commodore introduced the CBM 8032 with a build-in speaker, most PET/CBM users had to develop their own means of getting their computers to squeek, hum, whistle, and sing. They came up with the idea of using the shift register of the 6522 connected to the user port to send square waves through an external amplifier/speaker combination. The shift register could be programmed through BASIC, giving a wide variety of squeals, pops, sirens, etc.

Theory
Most music is made up of square waves of different amplitudes and frequencies. One of the functions of the 6522 chip is to generate square waves through the CB2 line. If we connect the CB2 line to a speaker, we will be able to hear the square waves generated by the VIC.

NOTE: Connecting a speaker directly to CB2 may damage your VIC and void your warranty. You must connect the speaker through an *amplifier* to protect the VIC.

Parts Needed

1. Small battery powered speaker/amplifier
2. User Port Connector (12 position, 24 contact edge connector with .145"spacing
3. Wire

Connecting The External Speaker to Your VIC

Bb	=251	(B below first C)	B	=124
C	=237	(first C)	C1	=117
C#	=224		C1#	=111
D	=211		D1	=104
D#	=199		D1#	= 99
E	=188		E1	= 93
F	=177		F1	= 88
F#	=167		F1#	= 83
G	=157		G1	= 78
G#	=149		G1#	= 73
A	=140		A1	= 69
A#	=132			

1. Wire the GROUND of the amplifier to the GROUND of the USER PORT (pin N) .
2. Wire the SIGNAL of the amplifier to the CB2 output of the USER PORT (pin M).

You are now ready to add your other voice through a BASIC program.

BASIC program steps:

1. Set the 6522 shift register to free running mode by typing:
   POKE 37147,16
2. Set the shift rate by typing:
   POKE 37144,C where C is an integer from 0 to 255
   C is the note to be played.
3. Load the shift register by typing:
   POKE37146,D where D = 15, 51, or 85 for a square wave. This step sets the octave for the note.

This step must be done last, since as soon as it is set, the VIC starts generating the square waves.

The frequency of the square wave can be found by the following formula:

$$\text{FREQUENCY} = \frac{500000\text{ Hz}}{(C+2)(D1)}$$

Where D1=8 when D=15
D1=4 when D=51
D1=2 when D=85

When you're in this mode, the VIC will *not read or write to cassette*. To restore normal operations, you must type:
POKE 37147,0

The following short program demonstrates music using this method. By hitting a letter a note will be played.

```
10 PRINT " MUSIC USING CB2."
11 REM A TO G IS ONE OCTAVE, SHIFT A TO G IS
   ANOTHER
15 PRINT "HIT + TO GO UP AN OCTAVE"
17 PRINT:PRINT "USE ! TO EXIT."
20 POKE3747,16:DIMA(14):FORI=1TO14:
   READA(I):NEXT
40 GETA$IFA$=""THEN40
42 IFA$="!" THEN POKE37147,0:END:REM
   RESET 6522
45 ifa$="+" THEN SF=SF-(SF<2):GOTO40
50 ifa$="+" THEN SF=SF-(SF>0):GOTO40
60 A=ASC(A$)-64+(ASC(A$)>192)*121:IF A>14
   OR A<1 THEN 40
70 POKE 37144,A(A)
80 POKE37146,-(SF=0)*15-(SF=1)*51-(SF=2)*85
90 GOTO40
100 DATA 124,117,104,93,88,78,69
110 DATA 251,237,211,188,177,157,140
```

One use for this procedure is to connect an external amplifier and speakers to your VIC to provide improved sound quality . . . or perhaps to use your VIC as a music synthesizer, with the proper program. This is only one of several hobbyist-type projects we will be describing in the VIC section of this magazine. Watch future issues for more hobby-related computer projects. ■

— *Andy Finkel*

Tiny-Aid For VIC-20

David A. Hook
Barrie, Ont.

Introduction

Since the early days of the PET, various enhancements for BASIC have been available—Toolkit and Power are two commercial examples. Bill Seiler, then of Commodore, produced the first public-domain version, called BASIC-Aid.

Many updates, corrections and improvements have been made over the past couple of years. The PET/CBM program has ballooned to a 4K package for almost every flavour of equipment configuration.

As has been customary in the Commodore community, Mr. Jim Butterfield developed a version of the BASIC-Aid. He called this TINYAID2 (or TINYAID4, for Basic 4.0). This offered the six most useful commands from the full-fledged program.

Following is my modification of that work, designed to provide VIC users with the same benefits. After using this for a while, I think you will find the added commands nearly indispensible.

Features

VIC Tiny Aid is a machine language program which consumes about 1200 bytes of your RAM memory. After you have loaded the program, type 'RUN' and hit 'RETURN'. The program repacks itself into high memory. The appropriate pointers are set so that BASIC will not clobber it. VIC Tiny Aid is now alive.

Once activated, five commands become attached to BASIC. They will function only in "direct" mode, i.e. don't include them in a program.

```
(1) NUMBER 1000,5          'RETURN'
    NUMBER 100,10
```

Renumbers a BASIC program with a given starting line number and given increment between line numbers. The maximum increment is 255.

All references after GOTO, THEN, GOSUB and RUN are automatically corrected. A display of these lines is presented on the screen as it works. If a GOTO refers to a non-existent line number, then it is changed to 65535. This is an illegal line number, and must be corrected before the BASIC program is used.

```
(2) DELETE 100-200         'RETURN'
    DELETE- 1500
    DELETE 5199 -
```

Deletes a range of lines from a BASIC program. Uses the same syntax as the LIST command, so any line-range may be specified for removal. DELETE with no range will perform like a NEW command, so be careful.

```
(3) FIND /PRINT/          'RETURN'
    FIND /A$/, 150-670
    FIND "PRINT", 2000-
```

Will locate any occurences of the characters between the "/" marks. Almost any character may mark the start/end of the string to be found, so long as both are the same. The first example will find all the PRINT instructions in the program.

If you are looking for a string of text which contains a BASIC keyword, you must use the quote characters as markers. This will prevent the search string from being "tokenized".

If a limited line-range is desired, use the same syntax as for LIST. Note that a comma (",") must separate the line-range from the end marker.

All lines containing the string are printed to the screen. If a line has more than one of them, each occurence will cause a repetiton of that line.

```
(4) CHANGE -PRINT-PRINT#4,-      'RETURN'
    CHANGE /ABC/XYZ/, 6000-
    CHANGE /DS$/D1$/, -5000
```

Using the same syntax as FIND, you may change any string to any other string in a BASIC program. This command is very powerful, and was not part of the early versions of Basic-Aid or Toolkit.

As before, you may indicate a line-range. As the changes are made, the revised lines are displayed on the screen.

Watch out for the difference between BASIC keywords and strings of text within quotes. You may use the quote characters to differentiate, as with FIND.

```
(5) KILL          'RETURN'
```

This command disables VIC Tiny Aid and its associated commands. A syntax error will be the result if any of the above commands are now tried.

Since the routine is safe from interference from BASIC, you may leave it active for as long as your machine stays on. It is possible that VIC Tiny Aid may interfere with other programs that modify BASIC's internal 'CHRGOT' routine. The KILL command allows you to avoid this conflict.

Procedure

The VIC contains no internal machine language monitor, which is really the only practical way to enter this program. So follow one of the three methods below to perform the task.

(1) Borrow an Upgrade (2.0) or Basic 4.0 PET/CBM, with its internal ML monitor. This will be the easiest method to work with the program included.

(2) Use your VIC-20, but you must have a machine language monitor:

-Jim Butterfield's TINYMON FOR VIC (Compute#20, January 1982). -my adaption of SUPERMON FOR VIC (The Transactor, Volume 3, Issue #5). -VICMON cartridge from Commodore.

(3) The easy way (?). Send $3, a blank cassette or 1540/2031/4040 diskette in a stamped, self-addressed mailer to me at:

58 Steel Street
BARRIE, Ontario, CANADA
L4M 2E9

Be sure its packaged securely. Diskettes will be returned in DOS 2.0 format. Only 2040 (DOS 1.0) owners need take extra care. (The programs need to be copied to a DOS 1.0 formatted disk. Don't SAVE or otherwise WRITE to the disk you get).

If you are using a VIC, and have a 3K RAM or SUPEREXPANDER cartridge, plug this in. It will be somewhat easier to follow, since programs are then "PET-compatible" without further juggling. However don't use the 8K or 16K expansion for this job.

If you are familiar with the operation of the ML monitor, please skip ahead to the specifics below.

You are about to type in about 2500 characters worth of "hexadecimal" numbers. In addition to the digits from zero to nine, the alphabetic characters from A-F represent numbers from ten to fifteen. These characters, and three instructions, will be all that are used to enter our program. You don't have to understand the process—just type in the characters exactly. It's not very exciting, but don't be too intimidated by the "funny" display.

Enter the machine language monitor program :

TINYMON/SUPERMON FOR VIC — LOAD and RUN the program.
PET/CBM — Type "SYS1024" and hit "RETURN".
VICMON Cartridge — "SYS 6*4096" or "SYS 10*4096" (depending on version you have), then type "RETURN".

NOTE: If you are working on the unexpanded VIC you will need to follow the alternate instructions in parentheses.

The cursor will be flashing next to a period character ("."). Type the entry starting at the current cursor position:

```
.M 0580 05C0     "RETURN"     (.M 1180 11C0)
```

Several lines should appear on the screen, much like the "memory-dump" which accompanies this article. A four-"digit" quantity called an "address" leads off a line, and either eight or five columns of two- "digit" values appear alongside.

Look at the tables of values in the article. They show eight rows of these addresses. Note that the first "block" has the address "0580", which matches the first address just above. The first row of the next table shows "05C0", which is the second (or ending) address just above.

Your mission is to type in the matching values from the article, in place of the two-"digit" values you see on the screen.

Remember to hit "RETURN" at the end of each screen line, or the changes won't be made.

Double-check the values you've typed. It's not easy to find an error later on.

Look at the next block of values. Type in the start/end addresses to display:

```
.M 05C0 0600     "RETURN"     (.M 11C0 1200)
```

Type in the values required and go on with the rest of the blocks.

You will use addresses ranging from:

05xx–06xx–07xx–08xx–09xx–0Axx

as shown in the tables. The "x" characters stand for the other two "digits" of the address in the leftmost column.

If you are working on the unexpanded VIC, the sequence of addresses is:

11xx–12xx–13xx–14xx–15xx–16xx

You will have to type these pairs of characters in place of the leading two shown just above.

With that task complete, we are ready to preserve this work on tape. So type:

```
.S"VIC AID.ML",01,0580,0AB6       "RETURN"

(or .S"VIC AID.ML",01,1180,16B6     "RETURN")
```

Mount a blank tape, and follow the instructions. Save a second copy, for safety.

Exit the ML monitor, with:

```
.X              "RETURN"
```

VERIFY the program normally before going any further.

Now comes the easy part. Type "NEW", then the BASIC listing. Enter this exactly, without including any extra text. Save this as "VIC AID.BAS" and VERIFY it.

Finally, LOAD "VIC AID.ML" and SAVE "VIC AID.REL" on another blank tape. Both the BASIC part and the machine language part have been SAVEd together.

Check-Out

We are going to check out the machine language using a "checksum" method. Type in "NEW" before proceeding. Now enter the following program:

```
10  i=0 : rem (i=3072 for unexpanded VIC)
20  t=0 : for j= 1408+i to 2741+i
30  t=t+peek(j)
40  next j
50  print t
```

After a few seconds, if the value 161705 appears, you've likely got it perfect. Go to the next section.

If not, there's at least one incorrect entry. Change the two values in Line 20, using the table below. Re-RUN the program and compare against the value in the third column.

Repeat the process for each row, noting any that don't match. Each row corresponds to two "blocks" from the last section. You will have to re-enter the ML monitor to re–check those sections that differ. Re-SAVE the ML part!!!

Block#	Value 1	Value 2	Checksum
1– 2	1408	1535	15201
3– 4	1536	1663	17221
5– 6	1664	1791	15925
7– 8	1792	1919	15117
9–10	1920	2047	15565
11–12	2048	2175	14141
13–14	2176	2303	15840
15–16	2304	2431	16276
17–18	2432	2559	15152
19–20	2560	2687	15194
21	2688	2741	6073

Operation

The final acid test. ReLOAD the program from tape and RUN it. The screen will clear and a brief summary of the added commands will be displayed. The cursor should return almost instantly, under the "READY." message.

If the cursor does not come back, there is something still amiss. All the values appearing in the article were produced from a working copy of the program (Honest!). You still have option (3) from the Procedure section available. If you do send a tape/disk now, include your non-functioning version. I can then do a compare, to see where the error(s) were.

This has been a massive exercise, and mistakes can easily creep in. Your comments are welcome.

```
 1  print "Sr vic tiny aid"
 2  print "q adapted for vic by:"
 3  print " david a. hook"
 4  print "q from 'tiny aid' by:"
 5  print " jim butterfield"
 6  print "q and 'basic aid' by:"
 7  print "bill seiler"
 8  print "qrsample commands:"
 9  print "qchange /?/print#4,/"
10  print "find .gosub., 200–"
11  print "delete 130–625"
12  print "number 100,5"
13  print "kill      (vic aid)"
14  sys (peek (43)+peek (44)*256+383)
```

```
.: 0580 a5 2d 85 22 a5 2e 85 23
.: 0588 a5 37 85 24 a5 38 85 25
.: 0590 a0 00 a5 22 d0 02 c6 23
.: 0598 c6 22 b1 22 d0 3c a5 22
.: 05a0 d0 02 c6 23 c6 22 b1 22
.: 05a8 f0 21 85 26 a5 22 d0 02
.: 05b0 c6 23 c6 22 b1 22 18 65
.: 05b8 24 aa a5 26 65 25 48 a5

.: 05c0 37 d0 02 c6 38 c6 37 68
.: 05c8 91 37 8a 48 a5 37 d0 02
.: 05d0 c6 38 c6 37 68 91 37 18
.: 05d8 90 b6 c9 df d0 ed a5 37
.: 05e0 85 33 a5 38 85 34 6c 37
.: 05e8 00 aa aa aa aa aa aa aa
.: 05f0 aa aa aa aa aa aa aa aa
.: 05f8 aa aa aa aa aa aa aa aa

.: 0600 df ad fe ff 00 85 37 ad
.: 0608 ff ff 00 85 38 a9 4c 85
.: 0610 7c ad d9 fb 00 85 7d ad
.: 0618 da fb 00 85 7e 4c 8f fc
.: 0620 00 f0 03 4c 08 cf a9 c9
.: 0628 85 7c a9 3a 85 7d a9 b0
.: 0630 85 7e 60 db fb 00 85 8b
.: 0638 86 97 ba bd 01 01 c9 8c

.: 0640 f0 10 d0 02 a4 8c a6 97
.: 0648 a5 8b c9 3a b0 03 4c 80
.: 0650 00 00 60 bd 02 01 c9 c4
.: 0658 d0 ed a5 8b 10 02 e6 7a
.: 0660 84 8c a2 00 00 86 a5 ca
.: 0668 e8 a4 7a b9 00 00 02 38
.: 0670 fd d9 ff 00 f0 13 c9 80
.: 0678 f0 13 e6 a5 e8 bd d8 ff

.: 0680 00 10 fa bd d9 ff 00 d0
.: 0688 e4 f0 bf e8 c8 d0 e0 84
.: 0690 7a a5 a5 0a aa bd f5 ff
.: 0698 00 48 bd f4 ff 00 48 20
.: 06a0 e9 fb 00 4c 73 00 00 20
.: 06a8 b2 fd 00 a5 5f a6 60 85
.: 06b0 24 86 25 20 13 c6 a5 5f
.: 06b8 a6 60 90 0a a0 01 b1 5f

.: 06c0 f0 04 aa 88 b1 5f 85 7a
.: 06c8 86 7b a5 24 38 e5 7a aa
.: 06d0 a5 25 e5 7b a8 b0 1e 8a
.: 06d8 18 65 2d 85 2d 98 65 2e
.: 06e0 85 2e a0 00 00 b1 7a 91
.: 06e8 24 c8 d0 f9 e6 7b e6 25
.: 06f0 a5 2e c5 25 b0 ef 20 33
.: 06f8 c5 a5 22 a6 23 18 69 02

.: 0700 85 2d 90 01 e8 86 2e 20
.: 0708 59 c6 4c 67 e4 20 7c c5
.: 0710 20 73 00 00 85 8b a2 00
.: 0718 00 86 49 20 8c fd 00 a5
.: 0720 a5 c9 00 00 d0 07 a2 02
.: 0728 86 49 20 8c fd 00 20 73
.: 0730 00 00 f0 03 20 fd ce 20
.: 0738 b2 fd 00 a5 5f a6 60 85

.: 0740 7a 86 7b 20 d7 ca d0 0b
.: 0748 c8 98 18 65 7a 85 7a 90
.: 0750 02 e6 7b 20 ca ff 00 f0
.: 0758 05 20 dc fd 00 b0 03 4c
.: 0760 8f fc 00 84 55 e6 55 a4
.: 0768 55 a6 31 a5 32 85 8b b1
.: 0770 7a f0 d8 dd 00 00 02 d0
.: 0778 ed e8 c8 c6 8b d0 f1 88

.: 0780 84 0b 84 97 a5 49 f0 5b
.: 0788 20 f0 fd 00 a5 34 38 e5
.: 0790 32 85 a7 f0 28 c8 f0 ca
.: 0798 b1 7a d0 f9 18 98 65 a7
.: 07a0 c9 02 90 40 c9 4b b0 3c
.: 07a8 a5 a7 10 02 c6 8b 18 65
.: 07b0 0b 85 97 b0 05 20 24 fe
.: 07b8 00 f0 03 20 0c fe 00 a5

.: 07c0 97 38 e5 34 a8 c8 a5 34
.: 07c8 f0 0f 85 8c a6 33 bd 00
.: 07d0 00 02 91 7a e8 c8 c6 8c
.: 07d8 d0 f5 18 a5 2d 65 a7 85
.: 07e0 2d a5 2e 65 8b 85 2e a5
.: 07e8 7a a6 7b 85 5f 86 60 a6
.: 07f0 43 a5 44 20 3d fe 00 20
.: 07f8 e1 ff a9 00 00 85 c6 a4

.: 0800 97 4c f2 fc 00 a4 7a c8
.: 0808 94 31 a9 00 00 95 32 b9
.: 0810 00 00 02 f0 15 c5 8b f0
.: 0818 05 f6 32 c8 d0 f2 84 7a
.: 0820 60 c9 ab f0 04 c9 2d d0
.: 0828 01 60 4c 08 cf 90 05 f0
.: 0830 03 20 a6 fd 00 20 6b c9
.: 0838 20 13 c6 20 79 00 00 f0

.: 0840 0b 20 a6 fd 00 20 73 00
.: 0848 00 20 6b c9 d0 e0 a5 14
.: 0850 05 15 d0 06 a9 ff 85 14
.: 0858 85 15 60 20 ca ff 00 85
.: 0860 43 20 ca ff 00 85 44 38
.: 0868 a5 14 e5 43 a5 15 e5 44
.: 0870 60 a5 7a 85 22 a5 7b 85
.: 0878 23 a5 2d 85 24 a5 2e 85

.: 0880 25 60 a5 22 c5 24 d0 04
.: 0888 a5 23 c5 25 60 a4 0b c8
.: 0890 b1 22 a4 97 c8 91 22 20
.: 0898 01 fe 00 d0 01 60 e6 22
.: 08a0 d0 ec e6 23 d0 e8 a4 0b
.: 08a8 b1 24 a4 97 91 24 20 01
.: 08b0 fe 00 d0 01 60 a5 24 d0
.: 08b8 02 c6 25 c6 24 4c 24 fe

.: 08c0 00 a0 00 00 84 a5 84 0f
.: 08c8 20 cd dd a9 20 a4 a5 29
.: 08d0 7f 20 d2 ff c9 22 d0 06
.: 08d8 a5 0f 49 ff 85 0f c8 b1
.: 08e0 5f f0 19 10 ec c9 ff f0
.: 08e8 e8 24 0f 30 e4 84 a5 20
.: 08f0 7c fe 00 c8 b1 ae 30 d6
.: 08f8 20 d2 ff d0 f6 20 d7 ca

.: 0900 38 60 a0 9d 84 ae a0 c0
.: 0908 84 af 38 e9 7f aa a0 00
.: 0910 00 ca f0 ee e6 ae d0 02
.: 0918 e6 af b1 ae 10 f6 30 f1
.: 0920 20 6b c9 a5 14 85 35 a5
.: 0928 15 85 36 20 fd ce 20 6b
.: 0930 c9 a5 14 85 33 a5 15 85
.: 0938 34 20 8e c6 20 ca ff 00

.: 0940 20 ca ff 00 d0 21 20 ac
.: 0948 ff 00 20 ca ff 00 20 ca
.: 0950 ff 00 d0 03 4c 8f fc 00
.: 0958 20 ca ff 00 a5 63 91 7a
.: 0960 20 ca ff 00 a5 62 91 7a
.: 0968 20 b7 ff 00 f0 e2 20 ca
.: 0970 ff 00 20 ca ff 00 20 ca
.: 0978 ff 00 c9 22 d0 0b 20 ca

.: 0980 ff 00 f0 c5 c9 22 d0 f7
.: 0988 f0 ee aa f0 bc 10 e9 a2
.: 0990 04 dd d4 ff 00 f0 05 ca
.: 0998 d0 f8 f0 dd a5 7a 85 3b
.: 09a0 a5 7b 85 3c 20 73 00 00
.: 09a8 b0 d3 20 6b c9 20 51 ff
.: 09b0 00 a5 3c 85 7b a5 3b 85
.: 09b8 7a a0 00 00 a2 00 00 bd

.: 09c0 00 00 01 c9 30 90 11 48
.: 09c8 20 73 00 00 90 03 20 82
.: 09d0 ff 00 68 a0 00 00 91 7a
.: 09d8 e8 d0 e8 20 73 00 00 b0
.: 09e0 08 20 91 ff 00 20 79 00
.: 09e8 00 90 f8 c9 2c f0 b8 d0
.: 09f0 96 20 ac ff 00 20 ca ff
.: 09f8 00 20 ca ff 00 d0 08 a9

.: 0a00 ff 85 63 85 62 30 0e 20
.: 0a08 ca ff 00 c5 14 d0 0f 20
.: 0a10 ca ff 00 c5 15 d0 0b 20
.: 0a18 d1 dd a9 20 4c d2 ff 20
.: 0a20 ca ff 00 20 b7 ff 00 f0
.: 0a28 d2 20 a2 ff 00 e6 97 20
.: 0a30 24 fe 00 e6 2d d0 02 e6
.: 0a38 2e 60 20 a2 ff 00 c6 97

.: 0a40 20 0c fe 00 a5 2d d0 02
.: 0a48 c6 2e c6 2d 60 20 f0 fd
.: 0a50 00 a0 00 00 84 0b 84 97
.: 0a58 60 a5 35 85 63 a5 36 85
.: 0a60 62 4c 8e c6 a5 63 18 65
.: 0a68 33 85 63 a5 62 65 34 85
.: 0a70 62 20 ca ff 00 d0 fb 60
.: 0a78 a0 00 00 e6 7a d0 02 e6

.: 0a80 7b b1 7a 60 89 8a 8d a7
.: 0a88 43 48 41 4e 47 c5 44 45
.: 0a90 4c 45 54 c5 46 49 4e c4
.: 0a98 4b 49 4c cc 4e 55 4d 42
.: 0aa0 45 d2 00 00 a5 fc 00 41
.: 0aa8 fc 00 a5 fc 00 c6 fb 00
.: 0ab0 98 fe 00 ac fb 00 aa aa
.: 0ab8 aa aa aa aa aa aa aa aa
```

RAM
PET
AB
E=MC2
CBM
DOF
FIG C
FIG B
FIG A
COMPUTER WRITEOUT
A
B
AB
ABA
ABAB
ABABA

COMMODORE 64K MEMORY EXPANSION BOARD

By Ira Neal

The Commodore 64 Memory Expansion Board adds 64K bytes of memory to the Commodore 8032, providing a total of 96K bytes of available RAM. The expansion memory board can be used with commercial software packages like Wordcraft, Wordcraft Ultra, Visicalc, and Silicon Office or with the extended BASIC language provided with the board.

The add-on memory is mapped into four 16K byte blocks. Only two of these blocks can reside in memory at one time. The expansion RAM is mapped in the same address space normally allocated for the system ROMs, I/O registers and screen memory (See Fig. 1). The expansion board is disabled at power up, so the 8032 will display 31743 bytes free. The expansion RAM is accessed by a write-only control register at location $FFF0 (See Fig. 2). The function of the control bits in this register are as follows:

CONTROL REGISTER BIT 0 – When equal to 1, addresses $8000 through $BFFF on the Expansion Memory Board are write protected. The screen is not protected if screen peek through is enabled. When equal to 0, the addresses are not write protected.

CONTROL REGISTER BIT 1 – When equal to 1, addresses $C000 though $FFFF on the Expansion Memory Board are write protected. The I/O registers are not write protected if I/O peek through is enabled. When equal to 0, the addresses are not protected.

CONTROL REGISTER BIT 2 – When equal to 1, block 1 is selected. When equal to 0, block 0 is selected. These blocks are 16K bytes and reside at locations $8000 through $BFFF.

CONTROL REGISTER BIT 3 – When equal to 1, block 3 is selected. When equal to 0, block 2 is selected. These blocks are 16K bytes and reside at locations $C000 through $FFFF.

CONTROL REGISTER BIT 4 – Reserved. No function.

CONTROL REGISTER BIT 5 – When equal to 1, screen peek through is enabled. This allows the screen memory to locations $8000 through $87FF to be accessed.

CONTROL REGISTER BIT 6 – When equal to 1, I/O peek through is enabled. This allows the I/O register at locations $E800 through $EFFF to be accessed.

CONTROL REGISTER BIT 7 – When equal to 1, enables the expansion memory and the above registers. Bit 7 defaults to 0 on power up.

A diskette containing programs for testing and controlling the expansion emory is supplied with the board. These include: '8032.MEM.PRG' a diagnostic test program, 'EXPANDED DEMO' a demonstration of the use of the expansion memory as a 'fast disk', 'EXPANDED-BASIC' a program to add expanded memory functions to Commodore BASIC, 'ADD-ON-MON' a TIM monitor with addition functions for the expanded memory, and 'ADD-ON-LOAD' a program that enables the loading of different operating systems.

EXPANDED-BASIC is a software routine that is loaded into high memory ($7800-$7BE0), leaving 29K bytes of contiguous BASIC program space in lower memory. This program is a psuedo-cache memory system, or 'fast-disk', that allows users to store or 'cache' programs and data in the expansion RAM area for ultra high-speed access. The following instructions are added to BASIC:

RECALL
Format: !r,0:"filename",s(u)(p),device
Purpose: Cache a file from disk.

LOAD
Format: !l,"program name"
Purpose: Move data from ADD-ON to BASIC text area.

OVERLAY
Format: !o,"program name"
Purpose: Overlays data from ADD-ON to current program in BASIC text area.

EXECUTE
Format: !e,"program name"
Purpose: Clears BASIC text area, Moves data from ADD-ON to BASIC text area and executes program.

QUIT
Format: !q
Purpose: Turn off expanded BASIC functions.

The programs and files are placed in the expansion memory in a contiguous manner. If data will not fit in the first expansion memory bank, wraparound will occur. All 64K bytes are available; however, the number of cached files is limited to ten.

ADD-ON-LOAD is a special loader which loads one of the three special versions of BASIC, provided on the supplied diskette. These BASIC systems are loaded into the expansion RAM to enable the execution of 40-column or 2.0 BASIC programs on the 8032. The supplied BASIC systems are: 'BASIC 2.0', a copy of 2.0 BASIC with a patch to re-initialise the 8032's screen controller; 'BASIC 4.0/40', a copy of the BASIC found in 4000 series machines; 'BASIC 4.0/80', a copy of the BASIC that is resident in the 8032.

In assembly language programming all 96K bytes of memory are available to the advanced assembly language programmer. They must remember that monitor calls can only be executed with the expansion RAM disabled, and the expansion RAM banks must be switched to access more than 64K bytes of contiguous memory.

The 64K Expansion Memory Board can be installed quickly with the supplied instructions. After installation is complete the correct operation can be verified by the execution of the 'EXPANDED DEMO' and '8032.MEM.PRG'. The 'EXPANDED DEMO' program caches four programs in the unexpanded RAM and executes them one at a time. The '8032.MEM.PRG' is a diagnostic test program for the expansion RAM.

µpd **MicroPro Design Pty. Ltd.**

Specialising in the sales & support of the Commodore PET/CBM Microcomputers, peripherals and interfaces, including:

- IEEE488-RS232 COMMUNICATIONS
- RS232/CENTRONICS PRINTER INTERFACE
- EPROM PROGRAMMER
- WORD PROCESSOR PRINTER INTERFACE

For full details and prices call or write:

205/6 Clarke St
CROWS NEST
Ph:(02) 438 1220

Postal: P.O. Box 153
NORTH SYDNEY
NSW 2060

SUBSCRIBING BY BANKCARD

Have the **Commodore Magazine** delivered to your address and **put it on Bankcard**
Special Subscription Rates ($25.00/1 yr.) for subscription received before Feb 21st – **SAVE $5.00**
Complete the following and post to:
MERVYN BEAMISH GRAPHICS PTY LTD
82 Alexander St, Crows Nest, NSW 2065

Bankcard no. ☐☐☐ ☐☐ ☐☐☐ ☐☐☐☐☐☐☐

Signature of Holder ____________ Expiry Date ____________

Address ____________

____

____________ Postcode ____________

Figure 1

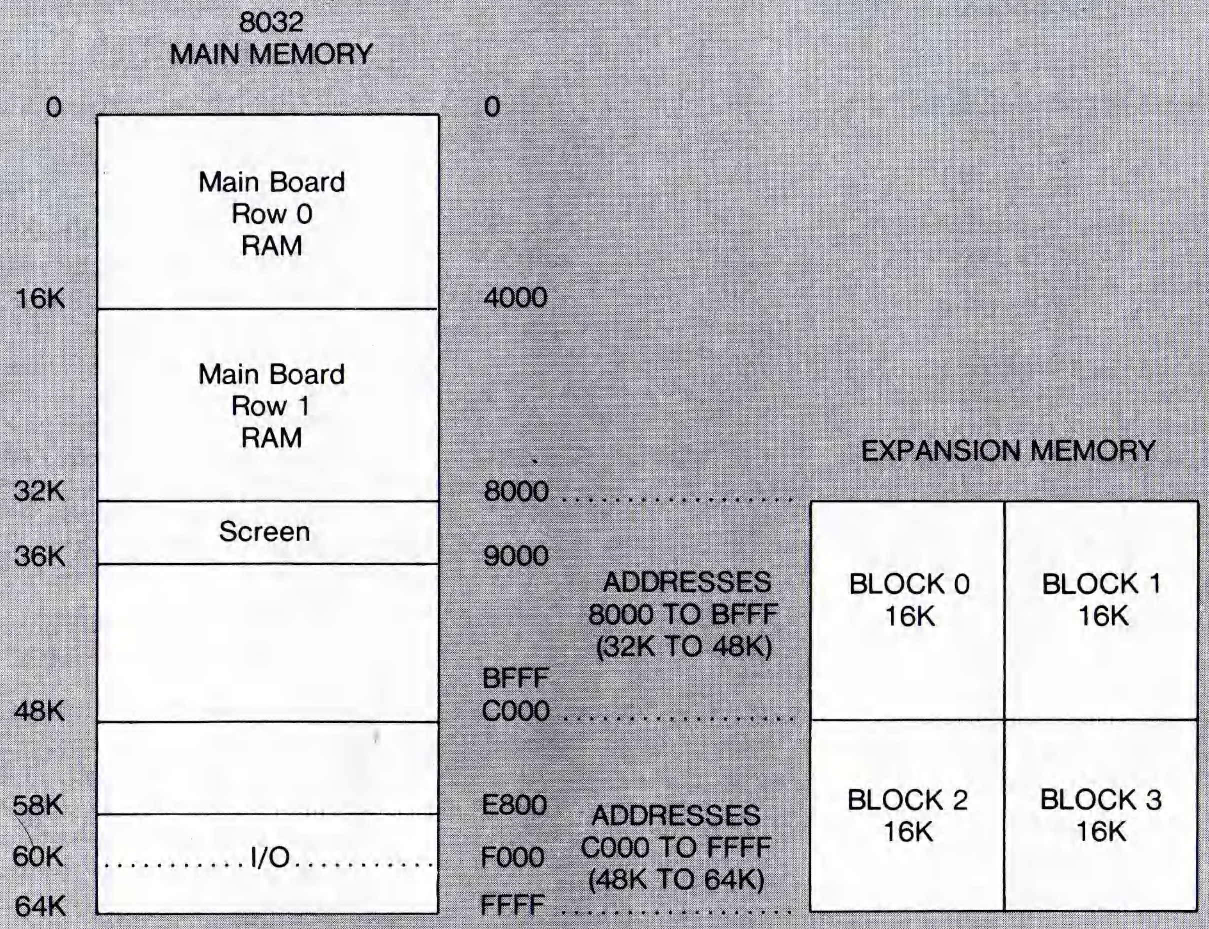
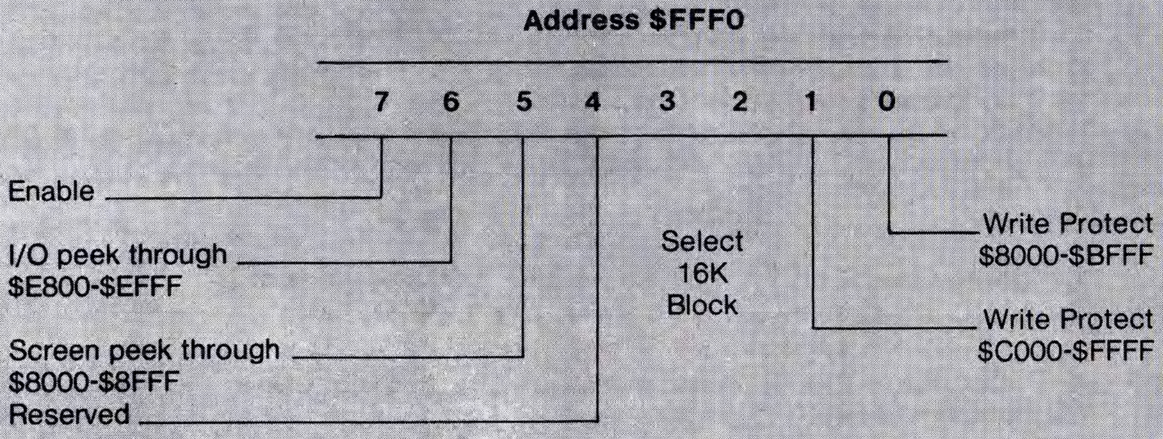

8032
MAIN MEMORY
0
16K
32K
36K
48K
58K
60K
64K
0
4000
8000
9000
BFFF
C000
E800
F000
FFFF
Main Board
Row 0
RAM
Main Board
Row 1
RAM
Screen
I/O
ADDRESSES
8000 TO BFFF
(32K TO 48K)
ADDRESSES
C000 TO FFFF
(48K TO 64K)
EXPANSION MEMORY
BLOCK 0
16K
BLOCK 1
16K
BLOCK 2
16K
BLOCK 3
16K

Figure 2


Address $FFF0
7 6 5 4 3 2 1 0
Enable
I/O peek through
$E800-$EFFF
Screen peek through
$8000-$8FFF
Reserved
Select
16K
Block
Write Protect
$8000-$BFFF
Write Protect
$C000-$FFFF

0 0	2 and 0
0 1	2 and 1
1 0	3 and 0
1 1	3 and 1

PROGRAMMER'S TIPS

Accessing the SuperPET RS-232 Port

Using the SuperPET's RS-232 serial port from the 6809 processor is made easy because of communication routines included in ROM and the SETUP menu which allows selection of stop-bit, parity and baud rate options. However, one of the great mysteries of the SuperPET has been how to access the RS-232 serial port using the 6502 processor. So, this month we will try to remove some of that mystery by providing details of register addresses and formats for programming the serial port.

The serial port on the SuperPET is a 6551 Asynchronous Communication Interface Adapter (ACIA) manufactured by Commodore Semiconductor Group. The ACIA uses a single +5 volt power supply, features an on-chip baud-rate generator and is capable of half-duplex or full-duplex operation. Word length, number of stop-bits, parity generation/checking and baud-rate are all programmable.

The ACIA is seen by both SuperPET microprocessors as four memory locations at address (hexadecimal) $EFF0–$EFF3. Input/output and programming of the ACIA is done by writing to or reading from these addresses as shown in Table 1.

commodore
SuperPET
computer

Table 1. SuperPET ACIA Memory Locations

Memory Address	WRITE Access	READ Access
$EFF0	Fill Transmitter Data Register	Unload Receiver Data Register
$EFF1	Programmed Reset (Use any data)	Read Status Register
$EFF2	Program Command Register	Read Command Register
$EFF3	Program Control Register	Read Control Register

Table 2. ACIA Control Register Programming

The Control Register is used to select the desired operating mode for the ACIA. The word-length, number of stop-bits, clock control and baud-rate are all programmed via the Control Register as shown in Table 2.

Control Bits	Control Function	Valid Data Values
7	Number of Stop Bits	0 = 1 Stop Bit 1 = 2 Stop Bits
6–5	Set Word Length (1.5 stop bits if 5 bits + Parity)	00 = 8 bits 01 = 7 bits 02 = 6 bits 03 = 5 bits
4	Select Clock Source (Always set to "1")	0 = External Clock 1 = Baud-rate Generator
3–0	Select Baud-rate	$0 (Hex) = Not Used $1 = 50 Baud $2 = 75 $3 = 110 $4 = 134.5 $5 = 150 $6 = 300 $7 = 600 $8 = 1200 $9 = 1800 $A = 2400 $B = 3600 $C = 4800 $D = 7200 $E = 9600 $F = 19200

Table 3. ACIA Command Register Programming

The Command Register in the 6551 ACIA is used to control parity generation/checking, receiver echo and transmit/receive functions as shown in Table 3.

Command Bits	Command Function	Valid Data Values
7–5	Set Parity Options	xx0 = Parity Disabled 001 = Odd Parity on Xmit & Recv 011 = Even Parity on Xmit & Recv 101 = Mark Parity Xmit Recv Parity Disabled 111 = Space Parity Xmit Recv Parity Disabled
4	Set Normal/Echo Mode	0 = Normal (No Echo) 1 = Echo for Receiver
3–2	Transmitter Control	00 = Xmitter Disabled, No Request-to-Send 01 = Xmitter Enabled, Request-to-Send 10 = Xmitter Disabled, Request-to-Send 11 = Xmitter Disabled, Request-to-Send (Transmit BRK)
1	Receiver Interrupt Enable	0 = Interrupt Enabled from Status Register Bit 0 1 = Interrupt Disabled
0	Data Terminal Ready	0 = Disable Recvr/Xmitter 1 = Enable Recvr/Xmitter

Table 4. ACIA Status Register Definitions

The Status Register is a read-only register which provides the processor with the status of various ACIA functions. The format of the Status Register is outlined in Table 4.

Status Bits	Status Functions and Values	
0*	1 = Parity Error Detected	0 = No Parity Error
1*	1 = Framing Error Detected	0 = No Framing Error
2*	1 = Overrun Has Occurred	0 = No Overrun
3	1 = Receiver Data Register is Full	0 = Receiver Not Full
4	1 = Transmitter Data Register is Empty	0 = Xmitter Not Empty
5	1 = No Data Carrier	0 = Carrier Detected
6	1 = Data Set Not Ready	0 = Data Set Ready
7	1 = Interrupt Requested	0 = No Interrupt Request
*	No Interrupt Request occurs for these status conditions ■	

— Dave Middleton

Weekday Calculator

This neat little subroutine returns the day of the week for any date given in DAY/MONTH/YEAR format. Of course you could change it around for YEAR/MONTH/DAY—just alter the order of the variables following the INPUT statement. The program does not check for date validity . . . but that's no problem. Just do some testing for day greater than 31 some months, 30 other months and 28 for February. For leap years, do an extra test of YEAR/4 = INT(YEAR/4) in the case of Feb. 29.

```
100 INPUT "DD, MM, YYYY";D,M,Y
110 K = INT( (60 + (100/M) )/100 )
120 F = 365 * Y + D + 31 * (M - 1) -
  INT(.4*M + 2.3) * (1 - K)
130 F = F + INT((Y - K/4) -
  INT(.75*(INT((Y - K) / 100 + 1))
140 F = F - INT(F/7) * 7
150 PRINT MID$
  ( "SATSUNMONTUEWEDTHUFRI",
  F * 3 + 1, 3 ) ■
```

```
90 PRINT" ENTER THE DATE INTHE
      FORMAT DD, MM, YYYY"
100 INPUT D,M,Y
```

SYS 'EM!

Two useful SYS addresses to note:

SYS 64790
SYS 54386

The first does a jump to 'warm start'—as if turning the machine off and back on again, but without that nasty power interruption. The second can be extremely handy when you want to send an M.L.M. memory dump to the printer. It seems that breaking to the monitor with SYS 4 cancels any CMD status you may have set up previously. ■

MAGAZINE SUBSCRIPTIONS

NAME ....

ADDRESS ....

....

SUBURB .... POSTCODE ....

COMMODORE
MAGAZINE
VOL 2
ISSUE 1
★ VIC Released
★ New CBM Software

Special Subscription Rate

(new or renewed subscritions
received prior to February 21st)

$A25.00 (Australia)
$A30.00 (Overseas)

Please enclose your remittance
with this renewal and send to:
COMMODORE MAGAZINE
P.O. BOX 336
ARTARMON
NSW 2064
AUSTRALIA

Please assist by completing
the following section:

I am a
VIC user ☐
CBM 8000 user ☐
PET 4000 user ☐

I am using
disk drives ☐
printers ☐
cassette ☐

I am
writing programs ☐
using standard programs ☐

What information would you like to see published?:

....

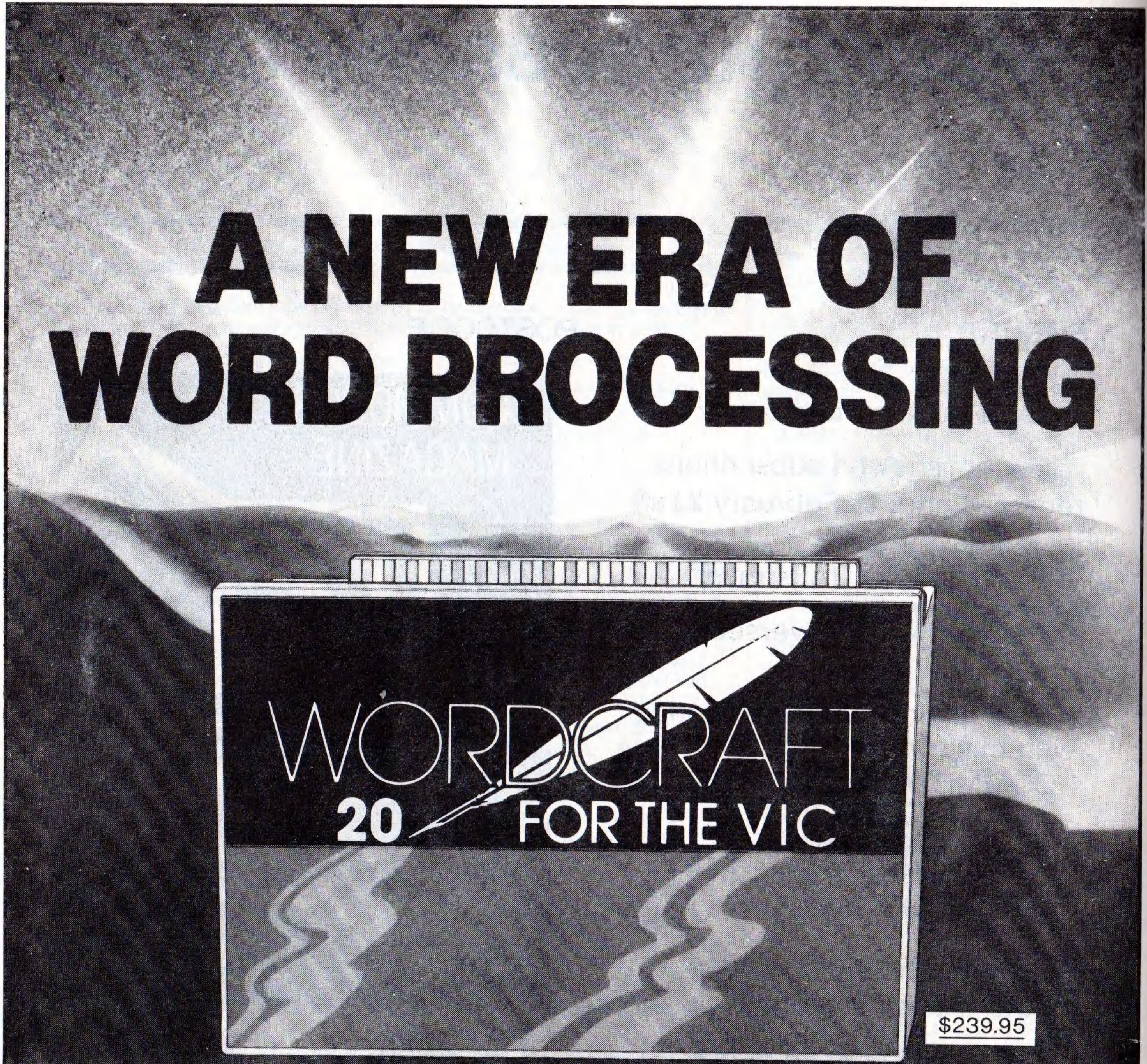
....

....

....

....

A NEW ERA OF WORD PROCESSING



The introduction of Wordcraft 20 for the VIC brings the benefits and advantages of full scale word processing directly to the general public.
Until now only the business world could afford word processing systems but this amazing price breakthrough makes it available to everyone.
Wordcraft 20 comes on a cartridge ready to plug into the back of the VIC. Included in the cartridge is an extra 8K of RAM that is also available for use with other programs – so not only do you get a word processor but you also get a memory expansion thrown in. The system also comes with complete documentation catering both for the inexperienced user and for those already familiar with Wordcraft 80.
Just look at these features:

- ★ Full use of colour and sound.
- ★ Full compatibility with VIC 1515 printer, parallel printers or RS232C serial printers.
- ★ Full control over margins, document width, tab stops, decimal tabs, justified output, multiple copies. Complete control of the final output.
- ★ Automatic underlining and emboldening.
- ★ Full screen display with automatic paging.
- ★ Full storage and retrieval facilities from disk and tape.
- ★ Full compatibility with Wordcraft 80.
- ★ Name and address capabilities – including labels.
- ★ Full document merging facilities.

Wordcraft 20. The package that the VIC user has been waiting for. A word processor of proven quality at a low price.
For the first time ever, every home can have one.

CW ELECTRONICS

416 LOGAN RD.(Pacific Hwy) STONES
CORNER, BRISBANE. TEL: (07) 397 0808,
397 0888 P.O. Box 274, SUNNYBANK QLD.4109. TELEX AA40811

THE VIC CENTRE

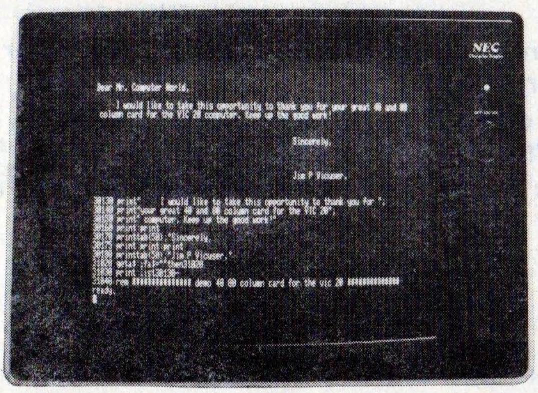
THE VIC CENTRE

VIC PLUS VIC PLUS

TELEPHONE DIALER

Turn your VIC 20 into a combination teledex and telephone dial. Just tell the VIC what name you want and presto! Other features included $119

VIC 20 40/80 COLUMN CARTRIDGE



Gives you 40-80 characters a column (not colour) with all VIC and CBM graphic characters. Whats more it plugs into a standard off-the-shelf VIC 20 – no expansion required. This little beauty will load all PET/CBM programmes and has all cursor controlled upper and lower case. The unit is switchable from basic with out losing program it requires no external power supply. Works well with 32k expansion if you wish. $259.95

NOTE: Our promotions manager is sorry but it seems that we've not been too clear on previous advertisments for this unit! – *THE VIC 20 40/80 CARTRIDGE WILL WORK ON A STANDARD OFF-THE-SHELF VIC 20.with no expansion required.*

INSTANT ROM EXPANSION CARTRIDGE

This cartridge allows you to expand to 32k with INSTANT ROM – *NO MOTHER BOARD REQUIRED -*

The cartridge this cartridge permitsothers to piggy back onto it giving these exclusive features ● Expansion to 32k with no mother board ● While using the cartridge slot it still permits use of other cartridges ● INSTANT ROM permits you to switch the VIC 20 off without losing its memory ● no additional power source required. $199

INSTANT ROM p.o.a.

EPROM PROGRAMMER

A must for the serious computerist and software developer. This unit includes all cables and interface program is in EPROM. $229

TDK-20 HAM INTERFACE

The TDK-20 is a complete RTTY and Morse code system for the VIC 20 computer. It is delivered in a single cartridge, which is plugged into a standard off-the-shelf VIC 20 or into an expansion board. Connect to your transciever (amatuer radio operators) or reciever (shortwave listeners) and away you go.

Features include ● on board converter ● LED tuning indicator ● All shifts available: 170,225, 425-850Hz ● instant break operation ● WRU

$229

Send for fuller details.

other optional extras include VIC CENTRE AFSK board	$ 49
12 vlt transformer	$6.50
User port plug (1 required)	$6.50

CW ELECTRONICS

DEALER ENQUIRIES WELCOME

Prices and specifications subject to change without notice

CANYON BOMBER

by Simon Holmes,
Maori Hill,
DUNEDIN,
NEW ZEALAND

This program was written on a 4032 computer but only uses 2K of memory.

The object is to drop bombs from the airship, using the space bar, and attempt to destroy the dots.

It is a copy of a relatively old arcade game of the same name.

```
READY.

 9 PRINT"◘":REM:    CANYON BOMBER
 10 :
 11 READA$:IFA$="*"THEN16
 12 PRINTTAB(20-(LEN(A$)/2));A$:GOTO11
 13 DATA"▓▓▓PLAY CANYON B0MBER","▓▓BY SIMON HOLMES","▓▓DROP BOMBS FROM AIRSHIP"
 14 DATA"▓USING SPACE","▓▓---------------------------","▓ HIT ▓[ SHIFT ]■ TO CONTINUE'
 15 DATA"▓▓","╭──╮","╰──╯"," ■","*"
 16 IFPEEK(152)=1THEN23
 17 X=32768:A=1:O=15
 18 X=X+A:POKEX,O:FORI=1TO10:IFPEEK(152)=1THEN23
 19 NEXT:POKEX,32:IFX=32790THENA=40
 20 IFPEEK(X+A)<>32THEN22
 21 GOTO18
 22 POKEX+A,15
 23 IFPEEK(152)=0THEN23
 99 :
 100 PRINT"◘":CLR
 110 POKE59467,16:POKE59466,1:POKE59464,0
 120 PRINT"◙":FORI=1TO10:PRINT:NEXT
 130 PRINT"▒▒▒▒▒●●●●●●●●●●●●●●●●●●●●●●●●●▒▒▒▒▒▒▒▒▒"
 140 PRINT"▒▒▒▒▒▒●●●●●●●●●●●●●●●●●●●●●●●▒▒▒▒▒▒▒▒▒▒"
 150 PRINT"▒▒▒▒▒▒▒●●●●●●●●●●●●●●●●●●●●●▒▒▒▒▒▒▒▒▒▒▒"
 160 PRINT"▒▒▒▒▒▒▒▒●●●●●●●●●●●●●●●●●●●▒▒▒▒▒▒▒▒▒▒▒▒"
 170 PRINT"▒▒▒▒▒▒▒▒▒●●●●●●●●●●●●●●●●●▒▒▒▒▒▒▒▒▒▒▒▒▒"
 180 PRINT"▒▒▒▒▒▒▒▒▒▒●●●●●●●●●●●●●●●▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
 190 PRINT"▒▒▒▒▒▒▒▒▒▒▒●●●●●●●●●●●●●▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
 200 PRINT"▒▒▒▒▒▒▒▒▒▒▒▒●●●●●●●●●●●▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
 210 PRINT"▒▒▒▒▒▒▒▒▒▒▒▒▒●●●●●●●●●▒▒▓ MISSES:"M"▌  ■▒▒▒▒▒"
 220 PRINT"▒▒▒▒▒▒▒▒▒▒▒▒▒▒●●●●●●●▒▒▓ SCORE:     ■▒▒▒▒"
 230 PRINT"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒●●●●●▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
 240 PRINT"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
 250 A=.5:S=1:P=32850
 260 Z$="╭──╮▓▌▌▌▌╰──╯▓▌▌▌▀][▌▌▌▌▌▌":REM:AIRSHIP GRAPHIC
 270 Y$=" ── ▓▌▌▌▌ ── ▓▌▌▌  ][▌▌▌▌▌▌":REM:ERASE AIRSHIP
 275 :
 280 PRINT"◙"SPC((S))Z$:GETB$:IFB$=" "THEN1000
 290 S=S+A:P=P+A
 300 IFS<1ORS>36THENA=-A
 315 POKE59464,255:FORI=1TO2:NEXT:POKE59464,0
 320 PRINT"◙"SPC((S-A))Y$:GOTO280
 325 :
 1000 REM: BOMB DROPPING ROVERFLOWINE
 1010 HH=0:HH=INT(RND(0)*4)+4:H=0
 1020 SO=1:DC=SC:FORI=P+40TO33688STEP40
 1030 PE=PEEK(I+40):IFPE=102THENPOKEI,32:GOTO1080
 1040 IFPE=81THENSC=SC+1:H=H+1
 1045 PRINT"◙▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▓"SC
 1050 IFSC=203THEN3010
 1060 IFHH<HTHENPOKE59464,0:GOTO1080
```

```
1070 POKEI,46:SO=SO+10:POKE59464,SO:FORO=1TO10:POKEI,32:NEXTO,I
1080 IFDC=SCTHENM=M+1:POKE59464,250:FORYY=1TO100:NEXT:POKE59464,0
1090 PRINT"SQQQQQQQQQQQQQQQQQQQ]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]R"M
1100 POKE59464,0:IFM>5THEN2000
1110 GOTO280
1120 :
2000 REM: END MESSAGE ROVERFLOWINE
2010 M$="R SORRY, THATS SIX MISSES ■"
2020 PRINT"SQQQQQ"TAB(8)M$
2030 M$="HIT SHIFT FOR NEW GAME":PRINT"Q"SPC(9);M$:FORI=1TO10:GETA$:NEXT
2040 IFPEEK(152)=0THEN2040
2050 FORI=1TO10:GETA$:GOTO100
2055 :
3000 REM: WINNING MESSAGE
3010 FORI=255TO1STEP-5:POKE59466,I:POKE59464,I:NEXT:POKE59464,0
3020 M$="R WELL DONE!  YOU WON ■":FORI=1TOLEN(M$):PRINT"SQQQQ"LEFT$(M$,I)
3030 FORW=1TO25:NEXTW,I
3040 GOTO2030
READY.
```

Pittwater Computer

NOW AVAILABLE.....

22 CARTER RD.,
BROOKVALE (02) 939 6760

The Electronic Cash Book on both the 4000 Series and the 8000 Series Commodore Computers.

This programme is designed to exactly emulate the hand-written Cash Book but has the added features of keeping a running bank balance; automatic deductions of periodical payments -full details of these are kept on file; reconciliation with the bank statements and a printed list of unreconciled cheques.

It has full budgetry figures; automatic searches via cheque number, payee or amount; also full reporting functions with transaction listings, dissection summaries, and detailed dissection listings. Depending upon the version, there is up to 39 incoming dissections and 59 out going dissections.

We also advise that the 8000 Series will flow directly into the IMS/COMMODORE General Ledger, thus making it the ideal package for small businesses.

The retail price is $400.00 plus tax, where applicable. Please contact your local dealer or Pittwater Computer Sales

DOS 1.2 PROBLEMS

1. SAVE WITH REPLACE

This command sometimes fails but the cause is not really known. It usually only happens on disks which have a lot of files and not a great deal of free space. The effect is to chain into other programs or files on the disk.

2. RENAME

This fails occasionally even though the disk system gives the '00,ok,00,00' message, the cause once again is not really known but it may fail due to there being scratched entries on the disk or the number of disk entries is a multiple of eight (ie. a full block).

3. DUPLICATE

If a disk is removed while doing a duplicate there is a very good chance that it will be totally corrupted so make sure that disks to be duplicated are in the correct drive before starting the command.

4. WRITE PROTECT TAB

Writing to a disk with a write protect tab is attempted. Then when a command is later given to read the disk, even if it has a write protect tab on, at least one write will be made. The solution is to power the disk drive down completely and then start again.

5. SEQUENTIAL FILES

If a sequential file of 254 characters (or any multiple) is written to the disk then an extra carriage return is added to the end of the file.

6. BLOCK ALLOCATE

The best way to use this command is to convert numbers into strings and concatennate this to the command before sending the command down the error channel.

7. ILLEGAL TRACK AND SECTOR

If illegal track or sector command parameters are given to the block commands then partial overlaying of error messages results.

8. BLOCK FREE

If an unallocated block is freed, the block count is automatically incremented by one and thus an incorrect number of blocks free can be generated ie. more than 670! Validate will restore the correct number of blocks.

9. VALIDATE 1

If an error occurs while validation of a diskette is taking place, then the BAM will be left in an indeterminate state. Re-initialisation of the diskette is necessary in order to restore the disk.

10. VALIDATE 2

The validate command frees any sectors allocated for random access.

11. SAVE AND OPEN WITHOUT GIVING A DRIVE NUMBER

This causes partial updating on both drives, thus corrupting both BAMs. This bug is probably the cause of more disk corruption problems than all the others put together, and may actually be the cause of some failures such as save with replaces.

12. DOS HANDLING OF THE IEEE BUS

Occasionally during multiple 'GET' the disk unit transmits a data byte onto the bus, even when the PET has Attention high. This gives the appearance that the PET is sending a command to all other peripherals on the bus.

13. USING ASTERISK AS THE FILENAME

An asterisk may be used as the filenme to access the last LOADed or SAVEd program. If the last program was saved with replace, then the asterisk accesses the old version of the program (which has now been scratched from the directory) rather than the program which was just saved.

14. MEMORY READ

The byte returned by a memory read operation is not accompanied by a carriage return so use GET to access the character.

RELATIVE RECORDS BUG

This is a serious bug in the relative record system in both DOS 2.1 and 2.5. The bug only occurs when two files have been opened for reading and writing. The bug only shows at certain length records and at set distances through the file. The following example demonstrates the bug:-

```
 30 DOPEN #1,"KEYTEST",L13,DO
 40 FOR J=11 TO 50-100
 50 A$=STR$(J)+"++++++++++++++++++++++":A$---MID$(A$,2,13)
 70 RECORD#1,(J)
 80 PRINT#1,A$
 90 NEXT
100 DCLOSE#1
110 DOPEN#2,"FILETEST",L254,DO
120 FOR J=1 TO 50
130 B$=STR$(J)+"+++++++++++++++++++++++++++++++++"
140 REORD#2,(J):PRINT#2,B$:NEXT
150 DCLOSE#2
190 DOPEN#1,"KEYTEST",L13,DO
200 FOR J=1 TO 50:INPUT#1,A$:PRINTA$:NEXT
210 DCLOSE#1
220 DOPENŁ2,"FILETEST",L254,DO
230 FOR J=1 TO 50:INPUT#2,A$:PRINTA$:NEXT
240 DCLOSE#2
250 PRINT"PRESS A KEY
260 GETZ$:IFZ$=""GOTO260
280 DOPEN#1,"KEYTEST",L13,DO
290 DOPEN#2,"FILETEST",L254,DO
300 X=34
310 FOR J=1 TO X:INPUT#1,A$
320 PRINTA$:NEXT
330 RECORD#2,25
340 INPUT#2,B$
350 PRINTB$
370 FOR J=X TO 50
380 A$=LEFT$(A$,9)+"TEST"
390 RECORD#1,(J)
400 PRINT#1,A$
410 PRINTA$
420 INPUT#1,A$
430 NEXT
440 DCLOSE#1
450 DCLOSE#2
510 DOPEN#1,"KEYTEST"L13,DO
520 FOR J=1 TO 50: INPUT#1,A$,PRINTA$:NEXT
530 DCLOSE
```

The program sets up 2 files (30 – 250) with unique records. The first 34 records are read from 'Keytest' then a record is read from 'Filetest'. Now records on 'Keytest' are updated. Both files are then closed (280 - 450). When 'Keytest' is read again some of the updated records are unchanged. In this example, records 34 – 40 are the same a they were originally.

Thus it is not possibe to have two relative files open for reading/writing at the same time with any degree of certainty that records will be updated correctly.

There are three solutions to this:-

1. **Open and close each file before accessing another.**
2. **Thoroughly test the record length chosen to see that it does not cause the bug.**
3. **This solution has no reason for working but it cured the bug in the example program so try it at your own risk: When the files are opened in lines 280 and 290, positiion the record pointer at record number 1, read it into the PET, reset the record pointer to 1 and then write it out again. The file then reads and updates correctly. Do this for both files.**

The following changes are to an article on the Data Cassette in the last issue of the Commodore Magazine.

ISSUE 7, PAGE 11

PROGRAM 1

```
5 OPEN 1,1,1,"TESTFILE"
10 FOR C=1 TO 3
20 INPUTA$
30 A$=LEFT$(A$+" ",10)
40 PRINT#1,A$;
50 NEXTC
60 PRINT#1,CHR$(13);
70 CLOSE1
```

PROGRAM 2

```
5 OPEN1,1,0,"TESTFILE"
10 FOR C=1 TO 30:GET#1,B$:
   A$=A$+B$:NEXTC
30 CLOSE1
40 B$=LEFT$(A$,10)
50 C$=MID$(A$,10,10)
60 D$=RIGHT$(A$,10)
70 PRINT A$,B$,C$,D$
```

ISSUE 7, PAGE 12

PROGRAM 1

```
10 REM PROGRAM TO WRITE DATA
   ONTO THE TAPE IN BLOCKS OF
   191 BYTES
20 OPEN1,1,1,"TESTFILE"
30 FORC=1TO5
40 PRINT#1,"1234";CHR$(13);
   "FRED";CHR$(13);
50 NEXTC
70 CLOSE1:END
100 REM WRITE OUT BUFFER
    ONTO TAPE
110 POKE 166,191 :REM SET
    POINTER TO LIMIT -1
120 PRINT#1," ";:REM PUT THE
    192ND CHAR INTO BUFFER
130 POKE 166,0 :REM RESET
    POINTER FOR MORE DATA
140 RETURN
```

PROGRAM 2

```
10 REM PROGRAM TO WRITE DATA
   TOO THE TAPE IN BLOCKS OF
   191 BYTES
20 OPEN1,1,1,"TESTFILE"
30 FORC=1TO5
40 PRINT#1,"1234";CHR$(13);
   "FRED";CHR$(13);
50 GOSUB 100
60 NEXTC
70 CLOSE1:END
100 REM WRITE OUT BUFFER
    ONTO TAPE
110 POKE 166,191 :REM SET
    POINTER TO LIMIT -1
```

Keyed Random Access for the PET/CBM

by

Glen Pearce
Commodore Johannesburg

Since the advent of Relative Files and the large storage capacity of the CBM 8050 Disk, some form of 'K.R.A.' (Keyed Random Access) would be useful to make full use of these facilities. Here is a version that meets most of the specifications of K.R.A., but is relatively (excuse the pun!) easy to use. It works as follows:

An ordinary sequential file is used to store a 'key-file' of all records held within a system (e.g. Stock, Accounts, Clients, etc.). This key-file would normally contain the first 10 characters of a customer's name (Part #, Account #, etc.) followed by *the Relative Record Number* of the record containing the remaining data for that customer.

Now, all you have to do is search through this key-file until you find the record you're looking for; retrieve the relative record number and you have access to the main record. The only problem in doing this in BASIC is time—especially if you have 500 to 1000 records or more!

Here is a machine-code routine which will do the above significantly faster (it searches through 500 ten-character record keys in approximately 4 seconds). This routine may only be used with BASIC 4.0 and DOS 2.0. Here's how you use it:

The length of each record in the key-file (SEQ) is not important and it may contain any valid ASCII characters (for safety's sake, stick to alpha-numerics only). To separate the record-key from the associated relative record number, a delimiter must be used. In this case the delimiter is a '#' symbol. Therefore, a record in the SEQ key-file should look something like:

SMITH# 1234

The space between the delimiter and the rel/rec number is the sign of the number and can be suppressed if space-saving on the disk is necessary.

It is important that each record in the key-file be separated by a Carriage Return—CHR$(13). This shouldn't present any problem as the PET/CBM automatically sends this character after each PRINT# command.

The K.R.A. machine code program must be located at the top of memory and protected in the usual way:

```
POKE 53, 127:POKE 52, 0:CLR
```

. . . must be the first statement in your program.

This program also allows you to do a form of "pattern-matching." Say, for instance, you don't know the exact spelling of a record-key in the key-file. All you do is enter the first few characters of the record-key and allow the program to search for that. When a 'match' is found in the file, the attached rel/rec number will be returned. You could then retrieve that relative record and display it. If it is NOT the correct record, simply tell the program to continue searching the key-file until it finds another match and so on. If NO match is found, a relative record number of 0 (zero) will be returned by the K.R.A. routine.

Here is an example of a BASIC program using the routine:

```
100 A$ = "":A = 0:REM INITIALIZE VARIABLES
    BEFORE USING K.R.A.
110 INPUT "ENTER SEARCH-STRING";A$
120 DOPEN#2, "KEY-FILE":IF DS< >0 THEN
    PRINT DS$:STOP
130 SYS 32512, 2, A$, A
140 IF A = 0 THEN DCLOSE#2:STOP:REM NO
    MATCH
150 REM RETRIEVE THE ASSOCIATED RELATIVE
    RECORD
160 REM AT THIS STAGE, IF THE REL/REC IS NOT
    CORRECT
170 REM YOU COULD 'GOTO 130' TO LOOK FOR
    ANOTHER MATCH
```

Any string and numeric variable may be used, but should be declared before the SYS 32512 to the routine. (In the above example 'A$' would have been initialized by the INPUT statement.) The '2' used after the first comma in the SYS command is the logical file number used in the DOPEN statement. It is important to check the DISK STATUS word (DS) after opening the file.

Adding records to the key-file could be a problem once the file gets large. Make use of the APPEND# command in BASIC 4.0 to simply append new record-keys to the file.

Another suggestion is to have separate key-files. For alphabetic keys there would be 26 titled 'A' to 'Z'; for numeric keys, 10 labelled '0' to '9'; or combine for alphanumeric and have 36 separate key files. Now you could simply check the first character of the search string (i.e., LEFT$(A$,1)) and open that particular file. This would reduce your key-file size to approximately 100 records per file in a 2000 record system, thereby making your search times even faster! ➡

```
30    REM  ************************************************************************
40    REM  *                                                                      *
50    REM  *           BASIC LOADER FOR MACHINE CODE ISAM ROUTINE                 *
60    REM  *                  GLEN PEARCE 20/8/81                                 *
70    REM  *                                                                      *
80    REM  ************************************************************************
90    REM
100   POKE53,127:CLR:REM LOWER MEMTOP TO PROTECT PROGRAM
110   FOR I = 32512 TO 32767:READ J:POKE I, J:NEXT:END
200   DATA   32,  73, 127,  32,  45, 201, 165,  18, 240,   3
210   DATA   76,   0, 191, 165,  17, 133, 210,  32,  82, 127
220   DATA  166, 210,  32, 198, 255, 160,   0,  32, 228, 255
230   DATA  166, 150, 208,  66, 201,  13, 240, 243, 209,   1
240   DATA  208,  18, 200, 196,   0, 144, 236,  32, 228, 255
250   DATA  166, 150, 208,  46, 201,  35, 240,  90, 208, 243
260   DATA   32, 228, 255, 166, 150, 208,  33, 201,  13, 240
270   DATA  210, 208, 243,  32, 245, 190,  32, 152, 189, 160
280   DATA    0,  96,  32,  73, 127, 177,  68, 133,   0, 200
290   DATA  177,  68, 133,   1, 200, 177,  68, 133,   2,  96
300   DATA   32,  73, 127, 169,   0, 133,  95, 133,  96  133
310   DATA    7, 162, 144,  32, 122, 205, 160,   0, 165,  94
320   DATA  145,  68, 200, 165,  95,  41, 127, 145,  68, 200
330   DATA  165,  96, 145,  68, 200, 165,  97, 145,  68, 200
340   DATA  165,  98, 145,  68,  32, 204, 255,  96,  32,  73
350   DATA  127, 169,   0, 133,  95, 133,   7,  32, 195, 127
360   DATA  201,  13, 240,  23, 166, 150, 208, 188, 133,  96
370   DATA   32, 195, 127, 201,  13, 240,  10, 166, 150, 208
380   DATA  175,  32, 213, 127,  76, 170, 127, 162, 144,  32
390   DATA  122, 205,  76, 116, 127,  32, 228, 255 201,  13
400   DATA  240,  10, 201,  48, 144, 245, 201,  58, 176, 241
410   DATA   41,  15,  96, 133,   0, 165,  95,  72, 165,  96
420   DATA   72,   6,  96,  38,  95,   6,  96,  38,  95, 104
430   DATA  101,  96, 133,  96, 104, 101,  95, 133,  95,   6
440   DATA   96,  38,  95, 165,   0, 101,  96, 133,  96, 169
450   DATA    0, 101,  95, 133,  95,  96
```

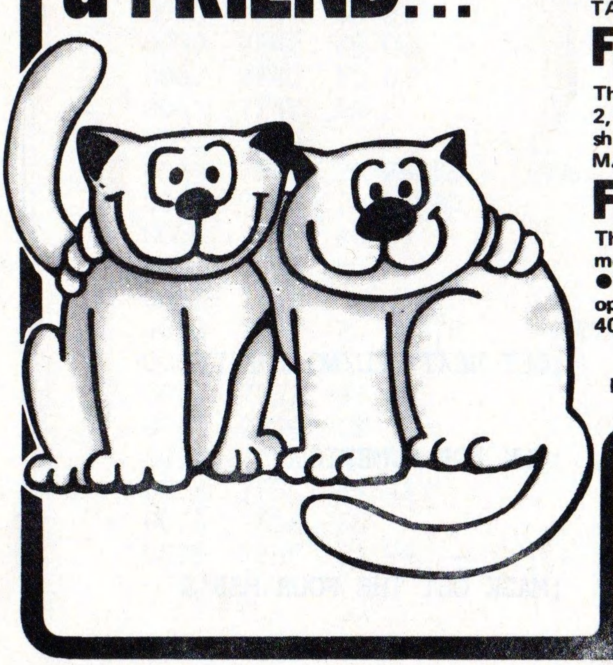
```
LINE# LOC CODE LINE

0001  0000            ;*************************************************
0002  0000            ;*  SEARCH THRU A SEQ FILE FOR A KEY RECORD AND  *
0003  0000            ;*   THEN RETRIEVE AN ATTACHED REL/REC NUMBER.   *
0004  0000            ;*                                               *
0005  0000            ;*    GLEN PEARCE                    22/08/81    *
0006  0000            ;*    COMMODORE, JOHANNESBURG,  SOUTH AFRICA     *
0007  0000            ;*************************************************
0008  0000            ;
0009  0000            ; ## CONSTANTS FROM PET BASIC (BASIC 4.0) ##
0010  0000            GETCHR = $FFE4         ;GET A CHARACTER
0011  0000            CLRCHN = $FFCC         ;CLOSE I/O CHANNELS
0012  0000            COIN   = $FFC6         ;SET INPUT DEVICE
0013  0000            CHKCOM = $BEF5         ;CHK FOR COMMA
0014  0000            FRMEVL = $BD98         ;EVALUATE EXPRESSION
0015  0000            FACINT = $C92D         ;CONVERT FL/P TO INT
0016  0000            SNERR  = $BF00         ;PRINT SYNTAX ERROR
0017  0000            ;
0018  0000            ; ## PAGE ZERO VARIABLES ##
```

```
0000                  LENGTH = $00         ;TEMP STORE OF STR LENGTH
0000                  WORK1  = $01         ;TEMP WORK AREA
0000                  CHKINT = $11         ;CHECK FOR INTEGER
0000                  CURFIL = $D2         ;CURRENT FILE NUMBER
0000                  VARPNT = $44         ; PNTR TO CURRENT VARIABLE
0000                  FAC    = $5E         ;MAIN FLT/PNT ACCUMULATOR
0000                  ;
0000                         * = $7F00
7F00                  ;
7F00  20 49 7F        FIND   JSR EVALEX    ;CHK SYNTAX OF COMMAND
7F03  20 2D C9               JSR FACINT    ;IN BASIC LINE & EXTRACT LFN
7F06  A5 12                  LDA CHKINT+1  ;AND SEARCH STRING
7F08  F0 03                  BEQ ISINTG
7F0A  4C 00 BF               JMP SNERR     ;EXIT IF SYNTAX ERROR
7F0D  A5 11           ISINTG LDA CHKINT
7F0F  85 D2                  STA CURFIL    ;SET UP LFN FOR READ
7F11  20 52 7F               JSR FNDEXP    ;FIND SRCH STRING
7F14  A6 D2                  LDX CURFIL
7F16  20 C6 FF               JSR COIN      ;SET I/O FOR READ
7F19                  ;
7F19  A0 00           GET10  LDY #0
7F1B  20 E4 FF        GET11  JSR GETCHR    ;GET CHAR FROM FILE
7F1E  A6 96                  LDX $96       ;CHK STATUS BYTE FOR EOF
7F20  D0 42                  BNE DONE1
7F22  C9 0D                  CMP #13       ;CHK FOR C/RET
7F24  F0 F3                  BEQ GET10     ;MOVE TO NEXT RECORD
7F26  D1 01                  CMP (WORK1)Y  ;COMPARE TO EQUIVALENT
7F28  D0 12                  BNE CLRSTR    ;CHAR OF SEARCH STRING
7F2A  C8                     INY
7F2B  C4 00                  CPY LENGTH    ;IF NUMBR OF CHARS CHK'D
7F2D  90 EC                  BCC GET11     ;EQUALS LEN OF SEARCH STRING
7F2F  20 E4 FF        FNDDEL JSR GETCHR    ;THEN MATCH IS MADE
7F32  A6 96                  LDX $96
7F34  D0 2E                  BNE DONE1
7F36  C9 23                  CMP #'#       ;FIND DELIMITER & THEN GO
7F38  F0 5A                  BEQ RELNUM    ;AND READ IN REL/NO.
7F3A  D0 F3                  BNE FNDDEL
7F3C  20 E4 FF        CLRSTR JSR GETCHR    ;DISCARD REST OF STRING
7F3F  A6 96                  LDX $96
7F41  D0 21                  BNE DONE1
7F43  C9 0D                  CMP #13
7F45  F0 D2                  BEQ GET10     ;GO AND CHK NEXT STRING
7F47  D0 F3                  BNE CLRSTR
7F49                  ;
7F49  20 F5 BE        EVALEX JSR CHKCOM    ;CHK FOR COMMA
7F4C  20 98 BD               JSR FRMEVL    ;& EVALUATE EXPRESSION
7F4F  A0 00                  LDY #0
7F51  60                     RTS
7F52                  ;
7F52  20 49 7F        FNDEXP JSR EVALEX    ;FIND SRCH STRING
7F55  B1 44                  LDA (VARPNT)Y ;SET UP STRING PNTRS
7F57  85 00                  STA LENGTH    ;IN TEMP WORK AREAS
7F59  C8                     INY
7F5A  B1 44                  LDA (VARPNT)Y
7F5C  85 01                  STA WORK1
7F5E  C8                     INY
7F5F  B1 44                  LDA (VARPNT)Y
```

```
0076  7F61  85 02             STA WORK1+1
0077  7F63  60                RTS
0078  7F64           ;
0079  7F64  20 49 7F  DONE1   JSR EVALEX       ;IF NO MATCH FOUND THEN
0080  7F67  A9 00             LDA #0           ;RETURN A REL/NO. OF ZERO
0081  7F69  85 5F             STA $5F
0082  7F6B  85 60             STA $60
0083  7F6D  85 07             STA $07          ;SET VARIABLE TYPE TO NUMERIC
0084  7F6F  A2 90             LDX #$90
0085  7F71  20 7A CD          JSR $CD7A        ;CONVERT HEX TO FL/P
0086  7F74  A0 00     DONE2   LDY #0
0087  7F76  A5 5E             LDA FAC          ;TRANSFER BCD VALUE OF
0088  7F78  91 44             STA (VARPNT)Y    ;REL/NO. TO NUMERIC VAR
0089  7F7A  C8                INY              ;SPECIFIED IN SYS CMD
0090  7F7B  A5 5F             LDA FAC+1
0091  7F7D  29 7F             AND #$7F         ;STRIP OFF SIGN
0092  7F7F  91 44             STA (VARPNT)Y
0093  7F81  C8                INY
0094  7F82  A5 60             LDA FAC+2
0095  7F84  91 44             STA (VARPNT)Y
0096  7F86  C8                INY
0097  7F87  A5 61             LDA FAC+3
0098  7F89  91 44             STA (VARPNT)Y
0099  7F8B  C8                INY
0100  7F8C  A5 62             LDA FAC+4
0101  7F8E  91 44             STA (VARPNT)Y
0102  7F90  20 CC FF          JSR CLRCHN       ;CLEAR ALL I/O CHANS AND
0103  7F93  60                RTS              ;EXIT PROGRAM
0104  7F94           ;
0105  7F94  20 49 7F  RELNUM  JSR EVALEX       ;FIND VARIABLE FOR REL/NO.
0106  7F97  A9 00             LDA #0
0107  7F99  85 5F             STA $5F
0108  7F9B  85 07             STA $07
0109  7F9D  20 C3 7F          JSR NEWDIG       ;READ IN REL/NO. AND CONVERT
0110  7FA0  C9 0D             CMP #13          ;IT TO A 2-BYTE HEX DIGIT
0111  7FA2  F0 17             BEQ PUTVAR
0112  7FA4  A6 96             LDX $96
0113  7FA6  D0 BC             BNE DONE1
0114  7FA8  85 60             STA $60
0115  7FAA  20 C3 7F  NXTDIG  JSR NEWDIG
0116  7FAD  C9 0D             CMP #13
0117  7FAF  F0 0A             BEQ PUTVAR
0118  7FB1  A6 96             LDX $96
0119  7FB3  D0 AF             BNE DONE1
0120  7FB5  20 D5 7F          JSR ASCHEX
0121  7FB8  4C AA 7F          JMP NXTDIG
0122  7FBB  A2 90     PUTVAR  LDX #$90
0123  7FBD  20 7A CD          JSR $CD7A
0124  7FC0  4C 74 7F          JMP DONE2
0125  7FC3           ;
0126  7FC3  20 E4 FF  NEWDIG  JSR GETCHR       ;GET NEXT REL/NO. DIGIT
0127  7FC6  C9 0D             CMP #13
0128  7FC8  F0 0A             BEQ ENDDIG
0129  7FCA  C9 30             CMP #$30         ;CHK FOR NUMERIC
0130  7FCC  90 F5             BCC NEWDIG
0131  7FCE  C9 3A             CMP #$3A
0132  7FD0  B0 F1             BCS NEWDIG
0133  7FD2  29 0F             AND #$0F         ;MASK OUT THE FOUR MSB'S
```

```
0134  7FD4  60          ENDDIG RTS
0135  7FD5              ;
0136  7FD5  85 00       ASCHEX STA LENGTH        ;HANDLE ASC - HEX CONVERSION
0137  7FD7  A5 5F              LDA $5F
0138  7FD9  48                 PHA
0139  7FDA  A5 60              LDA $60
0140  7FDC  48                 PHA
0141  7FDD  06 60              ASL $60
0142  7FDF  26 5F              ROL $5F
0143  7FE1  06 60              ASL $60
0144  7FE3  26 5F              ROL $5F
0145  7FE5  68                 PLA
0146  7FE6  65 60              ADC $60
0147  7FE8  85 60              STA $60
0148  7FEA  68                 PLA
0149  7FEB  65 5F              ADC $5F
0150  7FED  85 5F              STA $5F
0151  7FEF  06 60              ASL $60
0152  7FF1  26 5F              ROL $5F
0153  7FF3  A5 00              LDA LENGTH
0154  7FF5  65 60              ADC $60
0155  7FF7  85 60              STA $60
0156  7FF9  A9 00              LDA #0
0157  7FFB  65 5F              ADC $5F
0158  7FFD  85 5F              STA $5F
0159  7FFF  60          RETN   RTS
0160  8000                     .END
ERRORS = 0000
```



Every PET needs a FRIEND...

Introducing a new series of programs that are 'FRIENDLY' to the user and represent outstanding value for money. . . . and there will be more 'FRIENDS' coming

FRIEND 1 - Word Processor

An on-line ROM chip for 8, 16, and 32K machines. This Word Processor program has been written by professionals specially for The Microcomputer House. The program can be used with or without a printer and will be available shortly in disk and tape versions as well.

WP CHIP	$85
DISK	$70
TAPE	$60

FRIEND 2 - Mailing List

This is a dual disk based system for the 4000 series microcomputers. It caters for 2,100 records per data disk and offers sort and select facilities. It will also be available shortly for single disk systems.

MAILING LIST $85

FRIEND 3 - Data Handler

This is a ROM chip containing a machine language routine which allows the programmer to control screen input. ●Alpha Field Entry●Numeric Field Entry●Date Entry ●Disk Fastget ● Field Reverse ● Field Flash. Each of these functions have different options. FRIEND 3 is a derivative of our security ROM used by all our packages. 4000 series and 8000 series $85 each.

All programs come with a complete instruction manual.

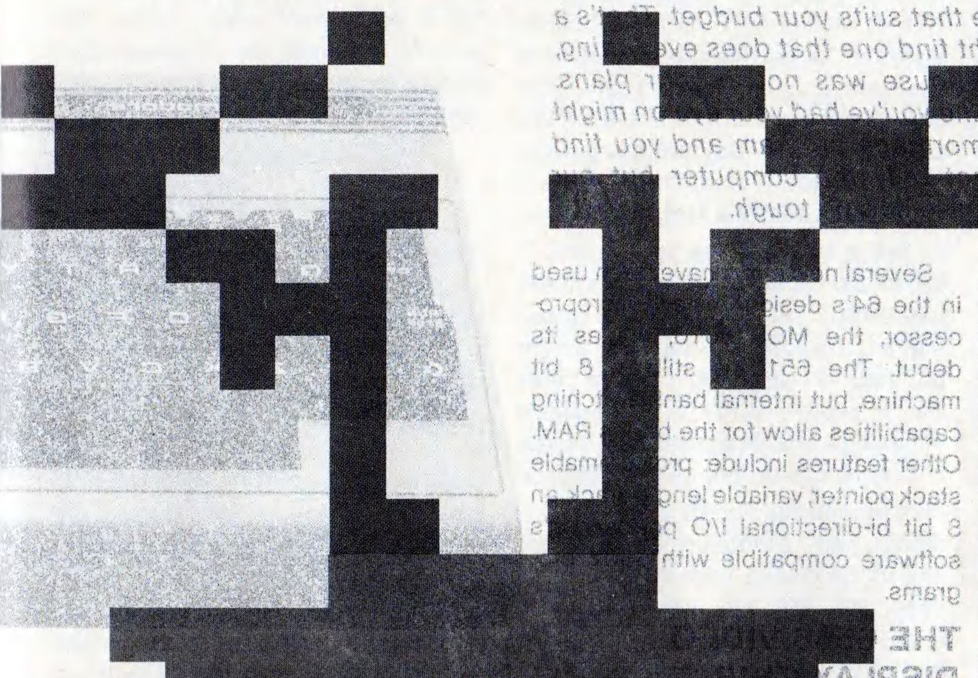
DEMONSTRATION STOCK AVAILABLE AT NEVER TO BE REPEATED PRICES
JUST PHONE OR CALL IN

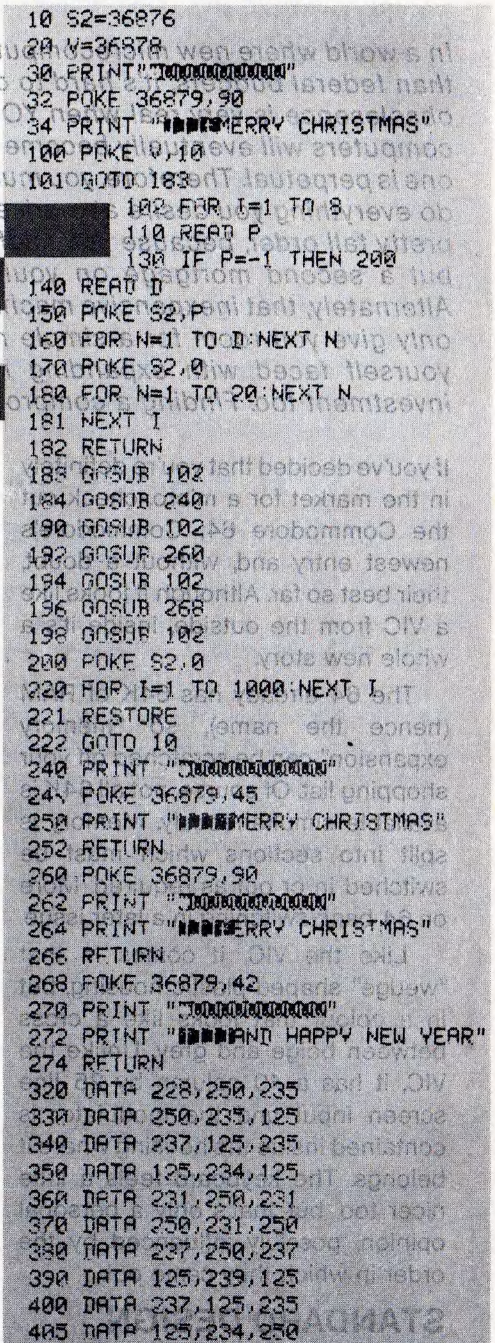
Bankcard Mail Orders Welcome

The Microcomputer House Pty. Ltd.

The VIC 20 Electronic Christmas Card

If you plan to give someone a VIC 20 this Christmas, this is a terrific little program you can include along with it. It plays "We Wish You a Merry Christmas" and displays the words to the song, while the screen changes into various red-green combinations.
Courtesy of Dr. Bruce Downing.





```
10 S2=36876
20 V=36878
30 PRINT"████████████"
32 POKE 36879,90
34 PRINT "████ERRY CHRISTMAS"
100 POKE V,10
101 GOTO 183
        102 FOR I=1 TO 8
        110 READ P
        130 IF P=-1 THEN 200
140 READ D
150 POKE S2,P
160 FOR N=1 TO D:NEXT N
170 POKE S2,0
180 FOR N=1 TO 20:NEXT N
181 NEXT I
182 RETURN
183 GOSUB 102
184 GOSUB 240
190 GOSUB 102
192 GOSUB 260
194 GOSUB 102
196 GOSUB 268
198 GOSUB 102
200 POKE S2,0
220 FOR I=1 TO 1000:NEXT I
221 RESTORE
222 GOTO 10
240 PRINT "████████████"
245 POKE 36879,45
250 PRINT "████MERRY CHRISTMAS"
252 RETURN
260 POKE 36879,90
262 PRINT "████████████"
264 PRINT "████ERRY CHRISTMAS"
266 RETURN
268 POKE 36879,42
270 PRINT "████████████"
272 PRINT "████AND HAPPY NEW YEAR"
274 RETURN
320 DATA 228,250,235
330 DATA 250,235,125
340 DATA 237,125,235
350 DATA 125,234,125
360 DATA 231,250,231
370 DATA 250,231,250
380 DATA 237,250,237
390 DATA 125,239,125
400 DATA 237,125,235
405 DATA 125,234,250
410 DATA 228,250,228
415 DATA 250,239,250
420 DATA 239,125,240
425 DATA 125,239,125
430 DATA 237,125,235
435 DATA 250,231,250
440 DATA 231,125,231
445 DATA 125,228,250
450 DATA 235,250,234
455 DATA 250,235,250
460 DATA -1

READY.
```

THE COMMODORE 64: A PRELIMINARY REVIEW

In a world where new microcomputers seem to be appearing faster than federal budgets, it's hard to decide when to buy. That fear of obsolesence is very real when YOUR money is on the line. But all computers will eventually become obsolete, so waiting for a better one is perpetual. Therefore, you must determine which computer will do everything you desire at a price that suits your budget. That's a pretty tall order, because you might find one that does everything, but a second mortgage on your house was not in your plans. Alternately, that inexpensive machine you've had your eye on might only give you room for a simple mortgage program and you find yourself faced with expanding not only the computer but our investment too. Finding a compromise can be tough.

If you've decided that you're definitely in the market for a micro, check out the Commodore 64; Commodore's newest entry and, without a doubt, their best so far. Although it looks like a VIC from the outside, inside it's a whole new story.

The 64 already has 64K of RAM (hence the name), so "memory expansion" can be scratched off your shopping list. Of course, not all 64K is available simultaneously. Memory is split into sections which must be switched in or out as required. More on 64 bank-switching in a later issue.

Like the VIC, it comes in that "wedge" shaped plastic housing, but in a colour that looks like a cross between beige and grey. Unlike the VIC, it has a 40 column by 25 line screen input and the modulator is contained inside the housing where it belongs. The keyboard feels a little nicer too, but that's only a personal opinion, possibly influenced by the order in which they came out.

STANDARD DESIGN FEATURES

★ Cartridge slot for games, etc.
★ 8 bit user port.
★ Serial bus for disk, printer, etc., (like on the VIC).
★ Cassette tape port.
★ Composite video output and modulator output ports.
★ Audio output jack.
★ 38K available for BASIC text.
★ 2 built-in Analog to Digital converters.
★ 16 colours.

And there's more...

Several new chips have been used in the 64's design. A new microprocessor, the MOS 6510, makes its debut. The 6510 is still an 8 bit machine, but internal bank switching capabilities allow for the bonus RAM. Other features include: programmable stack pointer, variable length stack, an 8 bit bi-directional I/O port and it's software compatible with 6502 programs.

THE 6566 VIDEO DISPLAY CHIP

This one's a beauty! Everything you ever wanted to do in movable graphics... and then some. The 6566 has the capability if moving definable shapes, or more commonly known as "sprites". Up until now, sprites have only been available on machines like arcade games. Pacmans, galaxians, frogs, etc, are all done using sprites.

The 64 allows 8 sprites to be displayed simultaneously (with more by using "raster compare" which we'll explain in a later issue). You simply define a shape somewhere in memory, select a sprite number, point a sprite at the shape, and turn it on. Like magic that shape will appear instantly on the screen. Then by merely giving X and Y coordinates, the shape moves around the screen by itself! No more erasing the shape at its previous location, no more updating screen RAM and colour tables... everything's done in hardware!

Other sprite features include: horizontal and/or vertical size expansion; hi-res or multi colour sprites; collision detection between sprites, or sprites and background; sprite/background display priority (sprites can appear in front of background or disappear behind background).

Other 6566 features include: smooth scrolling horizontally and vertically; 16 colours with 3 grey shades; a bit map mode for hi-resolution display; I/O ports for 2 joysticks or 4 paddles or light pen input.

THE 6581 SOUND INTERFACE DEVICE (SID)

The SID is virtually a synthesizer on a chip. It has 3 independenty controllable voices, each with a 9 octave range and 4 waveforms including square, triangle, sawtooth and noise. Each voice also has a programmable envelope generator and volume control with a master volume control for all three voices. The SID has some other very sophisticated features such as oscillator synchronization, ring modulation, filter resonance control, and it even has an external input for processing signals from other sources such as an electric guitar.

In all, the 6581 can produce sound with better quality than some of the instruments it can simulate. With a little programming, voice synthesis shouldn't be too much trouble either.

THE 6526 COMPLEX INTERFACE ADAPTER (CIA)

The C64 comes with 2 of these. The 6526's replace the 6520 and 6522 of earlier machines. Each chip has an 8 bit shift register for serial I/O, 24 clock with programmable alarm, 8 or 16 bit handshaking on read or write, 2 independent 16 bit interval timers and the capability for sourcing or sinking 2 standard TTL loads. I've been told that the external input to the SID can only process signals through the filter section which eliminates several possibilities. However, the 6526 has two analog to digital converters. By connecting external sources here, audio signals can be sent through all sections of the SID which will provide for some rather interesting experimentation. The A/D's will also eliminate a lot of the analog interfacing problems that have plagued us in the past.

ACCESSORIES

Commodore intends to use many of the same accessories for the 64 as are available now for the VIC. VIC Joysticks, paddles, disk drive, printer, modem, RS-232 cartridge and C2N cassette are all compatible with the 64. About the only difference is the slot for things like games and utility cartridges. It's been changed to a vertical pin type connector as opposed to the card edge type connector on the VIC 20. This is not only less space consuming, but promises to be a little more rugged as the contacts appear to be less susceptible to friction wear.

Future accessories, according to Commodore, include a Z-80 plug-in card, soft-load modules for BASIC 4.0, Pascal, Forth, and Pilot, extended BASIC cartridges for graphic and sound support, and monitor type cartridges for machine language exercises.

SOME COMMENTS

This time Commodore's done it right! They placed screen RAM immediately following zero page, the stack, and RAM allocated for the cassette buffer. This leaves a 38K stretch of uninterrupted memory which has been set up for BASIC text. Unlike the VIC, the screen won't be moving around you, but like the VIC, the LOAD command will adjust for the new start of text address, $0800.

The character generator is set up like in PET/CBMs. It takes no address space away from the processor unless you want to modify it. Using a very simple program, it can be transferred to RAM (where it will require address space) and a character pointer invokes the new location.

The 64 has 16 colours, 8 more than the VIC. Commodore had some trouble with colour quality on TV and monitor output, but this has been cleared up in the release versions. It has three distinct grey shades that come up beautifully even through modulated output to a TV. They also have an element of brilliance about them that make them appear more like colours as opposed to just shades.

The Control key is always a nice feature on any machine. The VIC Control key apparently had some problems when used in a communications environment, but this is all cleared up for the 64. RUN/STOP-RESTORE will still get you out of just about any crash, and changing from Upper/Lower case to Graphics mode is a snap with the shifted Commodore key at the lower left corner of the keyboard. 8 Function keys are included (4 plus 4 shifted) and I've been informed that a new method of keyboard scanning will allow the Control and Commodore keys to also be used as shift keys. Combinations of CTRL, Shift, and the Commodore key might result in as many as 32 effective function keys!

Commodore opted to stay with the serial bus for interfacing peripherals. Unlike IEEE parallel, bits are delivered in serial which reduces communications speed considerably. This reduction isn't really noticeable with the printer, but disk access is somewhat hampered. Fortunately there will be an IEEE interface card available for those that already own IEEE peripherals.

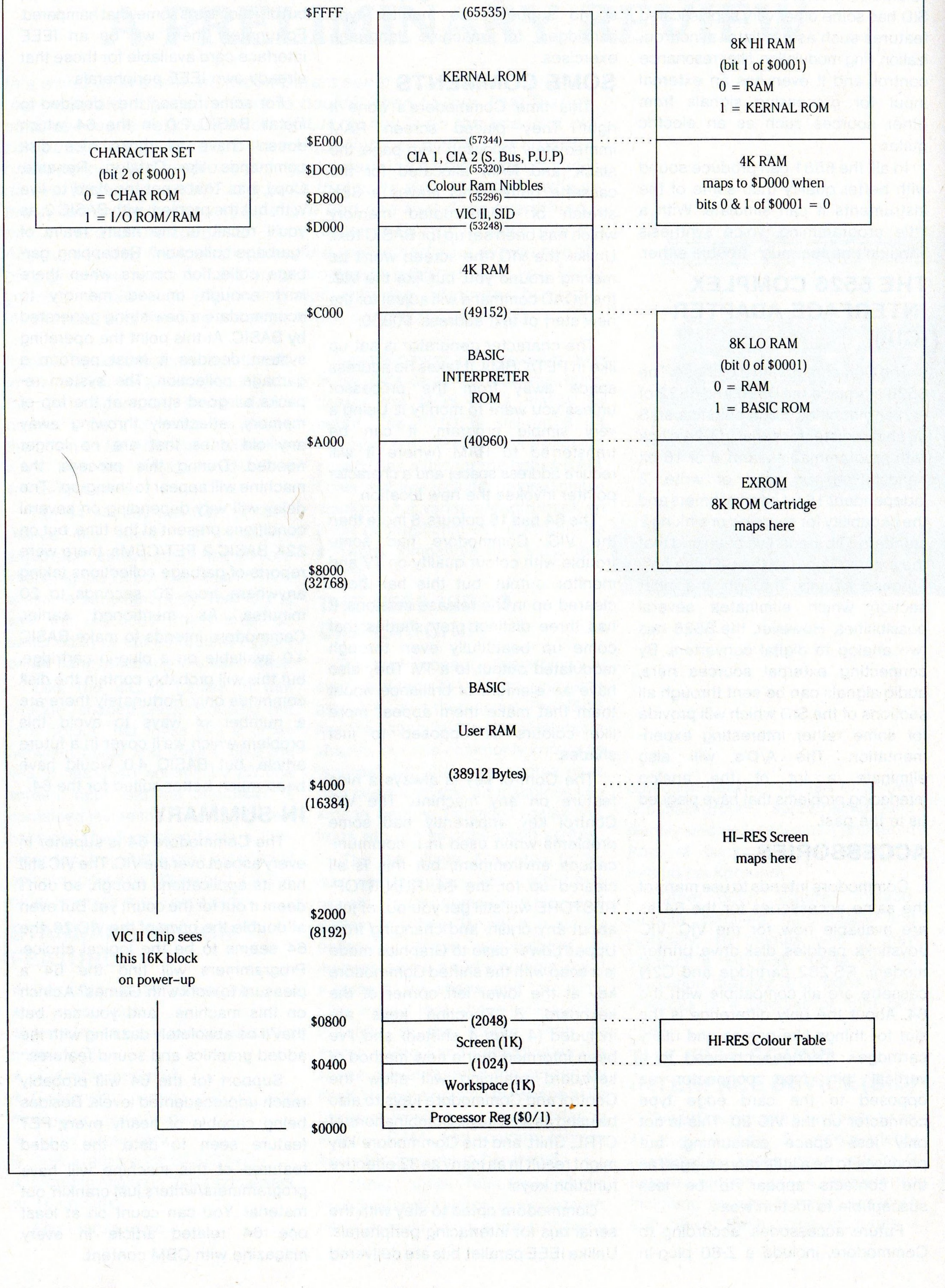
For some reason they decided to install BASIC 2.0 in the 64 which doesn't have all those nice disk commands like Catalog, Rename, Copy, etc. That's not too hard to live with, but the problem with BASIC 2, as you'll recall, is the nasty realm of "garbage collection". Recapping, garbage collection occurs when there isn't enough unused memory to acommodate a new string generated by BASIC. At this point the operating system decides it must perform a garbage collection. The system repacks all good strings at the top of memory, effectively throwing away any old ones that are no longer needed. During this process, the machine will appear to "hang-up". The delay will vary depending on several conditions present at the time, but on 32K BASIC 2 PET/CBMs, there were reports of garbage collections taking anywhere from 20 seconds to 20 minutes. As mentioned earlier, Commodore intends to make BASIC 4.0 available on a plug-in cartridge, but this will probably contain the disk commnds only. Fortunately there are a number of ways to avoid this problem which we'll cover in a future article, but BASIC 4.0 would have been much better suited for the 64.

IN SUMMARY

The Commodore 64 is superior in every aspect over the VIC. The VIC still has its applications though, so don't deem it out for the count yet. But even at double the price of the VIC 20, the 64 seems to be the logical choice. Programmers will find the 64 a pleasure to work with. Games? A cinch on this machine... and you can bet they'll be absolutely dazzling with the added graphics and sound features.

Support for the 64 will probably reach unprecedented levels. Besides being capable of nearly every PET feature seen to date, the added features of this machine will have programmers/writers just crankin' out material. You can count on at least one 64 related article in every magazine with CBM content.

Commodore–64 Architecture Map


$FFFF
(65535)
KERNAL ROM
8K HI RAM
(bit 1 of $0001)
0 = RAM
1 = KERNAL ROM
CHARACTER SET
(bit 2 of $0001)
0 = CHAR ROM
1 = I/O ROM/RAM
$E000
(57344)
4K RAM
maps to $D000 when
bits 0 & 1 of $0001 = 0
$DC00
CIA 1, CIA 2 (S. Bus, P.U.P)
(55320)
$D800
Colour Ram Nibbles
(55296)
VIC II, SID
$D000
(53248)
4K RAM
$C000
(49152)
BASIC
INTERPRETER
ROM
8K LO RAM
(bit 0 of $0001)
0 = RAM
1 = BASIC ROM
$A000
(40960)
EXROM
8K ROM Cartridge
maps here
$8000
(32768)
BASIC
User RAM
(38912 Bytes)
$4000
(16384)
HI-RES Screen
maps here
$2000
(8192)
VIC II Chip sees
this 16K block
on power-up
$0800
(2048)
Screen (1K)
HI-RES Colour Table
$0400
(1024)
Workspace (1K)
Processor Reg ($0/1)
$0000

Commodore 64 Memory Map

Compiled by
Jim Butterfield

0000		0	Chip directional register
0001		1	Chip I/O; memory & tape control
0003	-0004	3–4	Float–Fixed vector
0005	-0006	5–6	Fixed–Float vector
0007		7	Search character
0008		8	Scan–quotes flag
0009		9	TAB column save
000A		10	0 = LOAD, 1 = VERIFY
000B		11	Input buffer pointer/# subscrpt
000C		12	Default DIM flag
000D		13	Type: FF = string, 00 = numeric
000E		14	Type: 80 = integer, 00 = floating point
000F		15	DATA scan/LIST quote/memry flag
0010		16	Subscript/FNx flag
0011		17	0 = INPUT;$40 = GET;$98 = READ
0012		18	ATN sign/Comparison eval flag
0013		19	Current I/O prompt flag
0014	-0015	20–21	Integer value
0016		22	Pointer: temporary string stack
0017	-0018	23–24	Last temp string vector
0019	-0021	25–33	Stack for temporary strings
0022	-0025	34–37	Utility pointer area
0026	-002A	38–42	Product area for multiplication
002B	-002C	43–44	Pointer: Start–of–Basic
002D	-002E	45–46	Pointer: Start–of–Variables
002F	-0030	47–48	Pointer: Start–of–Arrays
0031	-0032	49–50	Pointer: End–of–Arrays
0033	-0034	51–52	Pointer: String–storage(moving down)
0035	-0036	53–54	Utility string pointer
0037	-0038	55–56	Pointer: Limit–of–memory
0039	-003A	57–58	Current Basic line number
003B	-003C	59–60	Previous Basic line number
003D	-003E	61–62	Pointer: Basic statement for CONT
003F	-0040	63–64	Current DATA line number
0041	-0042	65–66	Current DATA address
0043	-0044	67–68	Input vector
0045	-0046	69–70	Current variable name
0047	-0048	71–72	Current variable address
0049	-004A	73–74	Variable pointer for FOR/NEXT
004B	-004C	75–76	Y-save; op-save; Basic pointer save
004D		77	Comparison symbol accumulator
004E	-0053	78–83	Misc work area, pointers, etc
0054	-0056	84–86	Jump vector for functions
0057	-0060	87–96	Misc numeric work area
0061		97	Accum#1: Exponent
0062	-0065	98–101	Accum#1: Mantissa
0066		102	Accum#1: Sign
0067		103	Series evaluation constant pointer
0068		104	Accum#1 hi–order (overflow)
0069	-006E	105–110	Accum#2: Exponent, etc.
006F		111	Sign comparison, Acc#1 vs #2

0070	112	Accum#1 lo-order (rounding)
0071 -0072	113–114	Cassette buff len/Series pointer
0073 -008A	115–138	CHRGET subroutine; get Basic char
007A -007B	122–123	Basic pointer (within subrtn)
008B -008F	139–143	RND seed value
0090	144	Status word ST
0091	145	Keyswitch PIA: STOP and RVS flags
0092	146	Timing constant for tape
0093	147	Load = 0, Verify = 1
0094	148	Serial output: deferred char flag
0095	149	Serial deferred character
0096	150	Tape EOT received
0097	151	Register save
0098	152	How many open files
0099	153	Input device, normally 0
009A	154	Output CMD device, normally 3
009B	155	Tape character parity
009C	156	Byte-received flag
009D	157	Direct = $80/RUN = 0 output control
009E	158	Tp Pass 1 error log/char buffer
009F	159	Tp Pass 2 err log corrected
00A0 -00A2	160–162	Jiffy Clock HML
00A3	163	Serial bit count/EOI flag
00A4	164	Cycle count
00A5	165	Countdown,tape write/bit count
00A6	166	Tape buffer pointer
00A7	167	Tp Wrt ldr count/Rd pass/inbit
00A8	168	Tp Wrt new byte/Rd error/inbit cnt
00A9	169	Wrt start bit/Rd bit err/stbit
00AA	170	Tp Scan;Cnt;Ld;End/byte assy
00AB	171	Wr lead length/Rd checksum/parity
00AC -00AD	172–173	Pointer: tape bufr, scrolling
00AE -00AF	174–175	Tape end adds/End of program
00B0 -00B1	176–177	Tape timing constants
00B2 -00B3	178–179	Pntr: start of tape buffer
00B4	180	1 = Tp timer enabled; bit count
00B5	181	Tp EOT/RS232 next bit to send
00B6	182	Read character error/outbyte buf
00B7	183	# characters in file name
00B8	184	Current logical file
00B9	185	Current secndy address
00BA	186	Current device
00BB -00BC	187–188	Pointer to file name
00BD	189	Wr shift word/Rd input char
00BE	190	# blocks remaining to Wr/Rd
00BF	191	Serial word buffer
00C0	192	Tape motor interlock
00C1 -00C2	193–194	I/O start address
00C3 -00C4	195–196	Kernel setup pointer
00C5	197	Last key pressed
00C6	198	# chars in keybd buffer
00C7	199	Screen reverse flag
00C8	200	End-of-line for input pointer
00C9 -00CA	201–202	Input cursor log (row, column)
00CB	203	Which key: 64 if no key

00CC		204	0 = flash cursor
00CD		205	Cursor timing countdown
00CE		206	Character under cursor
00CF		207	Cursor in blink phase
00D0		208	Input from screen/from keyboard
00D1	-00D2	209–210	Pointer to screen line
00D3		211	Position of cursor on above line
00D4		212	0 = direct cursor, else programmed
00D5		213	Current screen line length
00D6		214	Row where curosr lives
00D7		215	Last inkey/checksum/buffer
00D8		216	# of INSERTs outstanding
00D9	-00F2	217–242	Screen line link table
00F3	-00F4	243–244	Screen color pointer
00F5	-00F6	245–246	Keyboard pointer
00F7	-00F8	247–248	RS–232 Rcv pntr
00F9	-00FA	249–250	RS–232 Tx pntr
00FF	-010A	256–266	Floating to ASCII work area
0100	-013E	256–318	Tape error log
0100	-01FF	256–511	Processor stack area
0200	-0258	512–600	Basic input buffer
0259	-0262	601–610	Logical file table
0263	-026C	611–620	Device # table
026D	-0276	621–630	Sec Adds table
0277	-0280	631–640	Keybd buffer
0281	-0282	641–642	Start of Basic Memory
0283	-0284	643–644	Top of Basic Memory
0285		645	Serial bus timeout flag
0286		646	Current color code
0287		647	Color under cursor
0288		648	Screen memory page
0289		649	Max size of keybd buffer
028A		650	Repeat all keys
028B		651	Repeat speed counter
028C		652	Repeat delay counter
028D		653	Keyboard Shift/Control flag
028E		654	Last shift pattern
028F	-0290	655–656	Keyboard table setup pointer
0291		657	Keyboard shift mode
0292		658	0 = scroll enable
0293		659	RS–232 control reg
0294		660	RS–232 command reg
0295	-0296	661–662	Bit timing
0297		663	RS–232 status
0298		664	# bits to send
0299	-029A	665	RS–232 speed/code
029B		667	RS232 receive pointer
029C		668	RS232 input pointer
029D		669	RS232 transmit pointer
029E		670	RS232 output pointer
029F	-02A0	671–672	IRQ save during tape I/O
02A1		673	CIA 2 (NMI) Interrupt Control
02A2		674	CIA 1 Timer A control log
02A3		675	CIA 1 Interrupt Log
02A4		676	CIA 1 Timer A enabled flag

02A5	677	Screen row marker	
02C0 -02FE	704–766	(Sprite 11)	
0300 -0301	768–769	Error message link	
0302 -0303	770–771	Basic warm start link	
0304 -0305	772–773	Crunch Basic tokens link	
0306 -0307	774–775	Print tokens link	
0308 -0309	776–777	Start new Basic code link	
030A -030B	778–779	Get arithmetic element link	
030C	780	SYS A–reg save	
030D	781	SYS X–reg save	
030E	782	SYS Y–reg save	
030F	783	SYS status reg save	
0310 -0312	784–785	USR function jump	(B248)
0314 -0315	788–789	Hardware interrupt vector	(EA31)
0316 -0317	790–791	Break interrupt vector	(FE66)
0318 -0319	792–793	NMI interrupt vector	(FE47)
031A -031B	794–795	OPEN vector	(F34A)
031C -031D	796–797	CLOSE vector	(F291)
031E -031F	798–799	Set–input vector	(F20E)
0320 -0321	800–801	Set–output vector	(F250)
0322 -0323	802–803	Restore I/O vector	(F333)
0324 -0325	804–805	INPUT vector	(F157)
0326 -0327	806–807	Output vector	(F1CA)
0328 -0329	808–809	Test–STOP vector	(F6ED)
032A -032B	810–811	GET vector	(F13E)
032C -032D	812–813	Abort I/O vector	(F32F)
032E -032F	814–815	Warm start vector	(FE66)
0330 -0331	816–817	LOAD link	(F4A5)
0332 -0333	818–819	SAVE link	(F5ED)
033C -03FB	828–1019	Cassette buffer	
0340 -037E	832–894	(Sprite 13)	
0380 -03BE	896–958	(Sprite 14)	
03C0 -03FE	960–1022	(Sprite 15)	
0400 -07FF	1024–2047	Screen memory	
0800 -9FFF	2048–40959	Basic RAM memory	
8000 -9FFF	32768–40959	Alternate: ROM plug-in area	
A000 -BFFF	40960–49151	ROM: Basic	
A000 -BFFF	49060–59151	Alternate: RAM	
C000 -CFFF	49152–53247	RAM memory, including alternate	
D000 -D02E	53248–53294	Video Chip (6566)	
D400 -D41C	54272–54300	Sound Chip (6581 SID)	
D800 -DBFF	55296–56319	Color nybble memory	
DC00 -DC0F	56320–56335	Interface chip 1, IRQ (6526 CIA)	
DD00 -DD0F	56576–56591	Interface chip 2, NMI (6526 CIA)	
D000 -DFFF	53248–53294	Alternate: Character set	
E000 -FFFF	57344–65535	ROM: Operating System	
E000 -FFFF	57344–65535	Alternate: RAM	
FF81 -FFF5	65409–65525	Jump Table, Including:	
FFC6		– Set Input channel	
FFC9		– Set Output channel	
FFCC		– Restore default I/O channels	
FFCF		– INPUT	
FFD2		– PRINT	
FFE1		– Test Stop key	
FFE4		– GET	

Commodore 64 – ROM Memory Map

A000;	ROM control vectors
A00C;	Keyword action vectors
A052;	Function vectors
A080;	Operator vectors
A09E;	Keywords
A19E;	Error messages
A328;	Error message vectors
A365;	Misc messages
A38A;	Scan stack for FOR/GOSUB
A3B8;	Move memory
A3FB;	Check stack depth
A408;	Check memory space
A435;	'out of memory'
A437;	Error routine
A469;	BREAK entry
A474;	'ready.'
A480;	Ready for Basic
A49C;	Handle new line
A533;	Re–chain lines
A560;	Receive input line
A579;	Crunch tokens
A613;	Find Basic line
A642;	Perform [NEW]
A65E;	Perform [CLR]
A68E;	Back up text pointer
A69C;	Perform [LIST]
A742;	Perform [FOR]
A7ED;	Execute statement
A81D;	Perform [RESTORE]
A82C;	Break
A82F;	Perform [STOP]
A831;	Perform [END]
A857;	Perform [CONT]
A871;	Perform [RUN]
A883;	Perform [GOSUB]
A8A0;	Perform [GOTO]
A8D2;	Perform [RETURN]
A8F8;	Perform [DATA]
A906;	Scan for next statement
A928;	Perform [IF]
A93B;	Perform [REM]
A94B;	Perform [ON]
A96B;	Get fixed point number
A9A5;	Perform [LET]
AA80;	Perform [PRINT#]
AA86;	Perform [CMD]
AAA0;	Perform [PRINT]
AB1E;	Print string from (y.a)
AB3B;	Print format character
AB4D;	Bad input routine
AB7B;	Perform [GET]
ABA5;	Perform [INPUT#]
ABBF;	Perform [INPUT]
ABF9;	Prompt & input
AC06;	Perform [READ]
ACFC;	Input error messages
AD1E;	Perform [NEXT]
AD78;	Type match check
AD9E;	Evaluate expression
AEA8;	Constant – pi
AEF1;	Evaluate within brackets
AEF7;	')'
AEFF;	comma..
AF08;	Syntax error
AF14;	Check range
AF28;	Search for variable
AFA7;	Setup FN reference
AFE6;	Perform [OR]
AFE9;	Perform [AND]
B016;	Compare
B081;	Perform [DIM]
B08B;	Locate variable
B113;	Check alphabetic
B11D;	Create variable
B194;	Array pointer subrtine
B1A5;	Value 32768
B1B2;	Float–fixed
B1D1;	Set up array
B245;	'bad subscript'
B248;	'illegal quantity'
B34C;	Compute array size
B37D;	Perform [FRE]
B391;	Fix–float
B39E;	Perform [POS]
B3A6;	Check direct
B3B3;	Perform [DEF]
B3E1;	Check fn syntax
B3F4;	Perform [FN]
B465;	Perform [STR$]
B475;	Calculate string vector
B487;	Set up string
B4F4;	Make room for string
B526;	Garbage collection
B5BD;	Check salvageability
B606;	Collect string
B63D;	Concatenate
B67A;	Build string to memory
B6A3;	Discard unwanted string
B6DB;	Clean descriptor stack
B6EC;	Perform [CHR$]
B700;	Perform [LEFT$]
B72C;	Perform [RIGHT$]
B737;	Perform [MID$]
B761;	Pull string parameters
B77C;	Perform [LEN]
B782;	Exit string–mode
B78B;	Perform [ASC]
B79B;	Input byte paramter
B7AD;	Perform [VAL]
B7EB;	Parameters for POKE/WAIT
B7F7;	Float–fixed
B80D;	Perform [PEEK]
B824;	Perform [POKE]
B82D;	Perform [WAIT]

B849;	Add 0.5
B850;	Subtract–from
B853;	Perform [subtract]
B86A;	Perform [add]
B947;	Complement FAC#1
B97E;	'overflow'
B983;	Multiply by zero byte
B9EA;	Perform [LOG]
BA2B;	Perform [multiply]
BA59;	Multiply–a–bit
BA8C;	Memory to FAC#2
BAB7;	Adjust FAC#1/#2
BAD4;	Underflow/overflow
BAE2;	Multiply by 10
BAF9;	+ 10 in floating pt
BAFE;	Divide by 10
BB12;	Perform [divide]
BBA2;	Memory to FAC#1
BBC7;	FAC#1 to memory
BBFC;	FAC#2 to FAC#1
BC0C;	FAC#1 to FAC#2
BC1B;	Round FAC#1
BC2B;	Get sign
BC39;	Perform [SGN]
BC58;	Perform [ABS]
BC5B;	Compare FAC#1 to mem
BC9B;	Float–fixed
BCCC;	Perform [int]
BCF3;	String to FAC
BD7E;	Get ascii digit
BDC2;	Print 'IN..'
BDCD;	Print line number
BDDD;	Float to ascii
BF16;	Decimal constants
BF3A;	TI constants
BF71;	Perform [SQR]
BF7B;	Perform [power]
BFB4;	Perform [negative]
BFED;	Perform [EXP]
E043;	Series eval 1
E059;	Series eval 2
E097;	Perform [RND]
E0f9;	?? breakpoints ??
E12A;	Perform [SYS]
E156;	Perform [SAVE]
E165;	Perform [VERIFY]
E168;	Perform [LOAD]
E1BE;	Perform [OPEN]
E1C7;	Perform [CLOSE]
E1D4;	Parameters for LOAD/SAVE
E206;	Check default parameters
E20E;	Check for comma
E219;	Parameters for open/close
E264;	Perform [COS]
E26B;	Perform [SIN]
E2B4;	Perform [TAN]
E30E;	Perform [ATN]
E37B;	Warm restart
E394;	Initialize
E3A2;	CHRGET for zero page
E3BF;	Initialize Basic
E447;	Vectors for $300
E453;	Initialize vectors
E45F;	Power–up message
E500;	Get I/O address
E505;	Get screen size
E50A;	Put/get row/column
E518;	InitializeI/O
E544;	Clear screen
E566;	Home cursor
E56C;	Set screen pointers
E5A0;	Set I/O defaults
E5B4;	Input from keyboard
E632;	Input from screen
E684;	Quote test
E691;	Setup screen print
E6B6;	Advance cursor
E6ED;	Retreat cursor
E701;	Back into previous line
E716;	Output to screen
E87C;	Go to next line
E891;	Perform <return>
E8A1;	Check line decrement
E8B3;	Check line increment
E8CB;	Set color code
E8DA;	Color code table
E8EA;	Scroll screen
E965;	Open space on screen
E9C8;	Move a screen line
E9E0;	Synchronize color transfer
E9F0;	Set start–of–line
E9FF;	Clear screen line
EA13;	Print to screen
EA24;	Synchronize color pointer
EA31;	Interrupt – clock etc
EA87;	Read keyboard
EB79;	Keyboard select vectors
EB81;	Keyboard 1 – unshifted
EBC2;	Keyboard 2 – shifted
EC03;	Keyboard 3 – 'comm'
EC44;	Graphics/text contrl
EC4F;	Set graphics/text mode
EC78;	Keyboard 4
ECB9;	Video chip setup
ECE7;	Shift/run equivalent
ECF0;	Screen ln address low
ED09;	Send 'talk'
ED0C;	Send 'listen'
ED40;	Send to serial bus
EDB2;	Serial timeout
EDB9;	Send listen SA
EDBE;	Clear ATN
EDC7;	Send talk SA
EDCC;	Wait for clock
EDDD;	Send serial deferred
EDEF;	Send 'untalk'

EDFE;	Send 'unlisten'
EE13;	Receive from serial bus
EE85;	Serial clock on
EE8E;	Serial clock off
EE97;	Serial output '1'
EEA0;	Serial output '0'
EEA9;	Get serial in & clock
EEB3;	Delay 1 ms
EEBB;	RS-232 send
EF06;	Send new RS-232 byte
EF2E;	No-DSR error
EF31;	No-CTS error
EF3B;	Disable timer
EF4A;	Compute bit count
EF59;	RS232 receive
EF7E;	Setup to receive
EFC5;	Receive parity error
EFCA;	Receive overflow
EFCD;	Receive break
EFD0;	Framing error
EFE1;	Submit to RS232
F00D;	No-DSR error
F017;	Send to RS232 buffer
F04D;	Input from RS232
F086;	Get from RS232
F0A4;	Check serial bus idle
F0BD;	Messages
F12B;	Print if direct
F13E;	Get..
F14E;	..from RS232
F157;	Input
F199;	Get.. tape/serial/rs232
F1CA;	Output..
F1DD;	..to tape
F20E;	Set input device
F250;	Set output device
F291;	Close file
F30F;	Find file
F31F;	Set file values
F32F;	Abort all files
F333;	Restore default I/O
F34A;	Do file open
F3D5;	Send SA
F409;	Open RS232
F49E;	Load program
F5AF;	'searching'
F5C1;	Print filename
F5D2;	'loading/verifying'
F5DD;	Save program
F68F;	Print 'saving'
F69B;	Bump clock
F6BC;	Log PIA key reading
F6DD;	Get time
F6E4;	Set time
F6ED;	Check stop key
F6FB;	Output error messages
F72D;	Find any tape headr
F76A;	Write tape header
F7D0;	Get buffer address
F7D7;	Set buffer start/end pointers
F7EA;	Find specific header
F80D;	Bump tape pointer
F817;	'press play..'
F82E;	Check tape status
F838;	'press record..'
F841;	Initiate tape read
F864;	Initiate tape write
F875;	Common tape code
F8D0;	Check tape stop
F8E2;	Set read timing
F92C;	Read tape bits
FA60;	Store tape chars
FB8E;	Reset pointer
FB97;	New character setup
FBA6;	Send transition to tape
FBC8;	Write data to tape
FBCD;	IRQ entry point
FC57;	Write tape leader
FC93;	Restore normal IRQ
FCB8;	Set IRQ vector
FCCA;	Kill tape motor
FCD1;	Check r/w pointer
FCDB;	Bump r/w pointer
FCE2;	Power reset entry
FD02;	Check 8-rom
FD10;	8-rom mask
FD15;	Kernal reset
FD1A;	Kernal move
FD30;	Vectors
FD50;	Initialize system constnts
FD9B;	IRQ vectors
FDA3;	Initialize I/O
FDDD;	Enable timer
FDF9;	Save filename data
FE00;	Save file details
FE07;	Get status
FE18;	Flag status
FE1C;	Set status
FE21;	Set timeout
FE25;	Read/set top of memory
FE27;	Read top of memory
FE2D;	Set top of memory
FE34;	Read/set bottom of memory
FE43;	NMI entry
FE66;	Warm start
FEB6;	Reset IRQ & exit
FEBC;	Interrupt exit
FEC2;	RS-232 timing table
FED6;	NMI RS-232 in
FF07;	NMI RS-232 out
FF43;	Fake IRQ
FF48;	IRQ entry
FF81;	Jumbo jump table
FFFA;	Hardware vectors

Processor I/O Port (6510)

$0000	IN	IN	OUT	IN	OUT	OUT	OUT	OUT	DDR 0
$0001			Tape Motor	Tape Sense	Tape Write	D-ROM Switch	EF RAM Switch	AB RAM Switch	PR 1

SID (6581)

Voice 1	Voice 2	Voice 3			Voice 1	Voice 2	Voice 3
$D400	$D407	$D40E	Frequency	L	54272	54279	54286
$D401	$D408	$D40F		H	54273	54280	54287
$D402	$D409	$D410	Pulse Width	L	54274	54281	54288
$D403	$D40A	$D411	0 0 0 0	H	54275	54282	54289
$D404	$D40B	$D412	NSE, Voice Type: PUL, SAW, TRI	Key	54276	54283	54290
$D405	$D40C	$D413	Attack Time 2ms – 8ms	Decay Time 6ms – 24 sec	54277	54284	54291
$D406	$D40D	$D414	Sustain Level	Release Time 6ms 24 sec	54278	54285	54292

Voices (write only)

$D415	0 0 0 0 0	L	54293
$D416	Filter Frequency	H	54294
$D417	Resonance	Ext, Filter Voices V3, V2, V1	54295
$D418	V3 off, Passband: HI, BP, LO	Master Volume	54296

Filter & Volume (write only)

$D419	Paddle X (A/D #1)	54297
$D41A	Paddle Y (A/D #2)	54298
$D41B	Noise 3 (random)	54299
$D41C	Envelope 3	54300

Sense (read only)

Note: Special Voice Features
(TEST, RING MOD, SYNC)
are omitted from the above diagram.

CIA 1 (IRQ) (6526)

$DC00	Paddle Sel A	B		Joystick 0 Fire	Right	Left	Down	Up	PRA	56320
	Keyboard Row Select (inverted)									
$DC01				Joystick 1 Fire	Right	Left	Down	Up	PRB	56321
	Keyboard Column Read									
$DC02	$FF – All Output								DDRA	56322
$DC03	$00 – All Input								DDRB	56323
$DC04	Timer A								TAL	56324
$DC05									TAH	56325
$DC06	Timer B								TBL	56326
$DC07									TBH	56327
$DC0D				Tape Input			Timer Interrupt B	A	ICR	56333
$DC0E					One Shot	Out Mode	Time PB6 Out	Timer A Start	CRA	56334
$DC0F					One Shot	Out Mode	Time PB7 Out	Timer B Start	CRB	56335

CIA 2 (NMI) (6526)

$DD00	Serial IN	Clock IN	Serial OUT	Clock OUT	ATN OUT	RS–232 OUT	VIC II addr 15	VIC II addr 14	PRA	56576
$DD01	DSR IN	CTS IN		DCD* IN	RI* IN	DTR OUT	RTS OUT	RS–232 IN	PRB	56577
$DD02	$3F – Serial								DDRA	56578
$DD03	$00 – P.U.P. All Input or $06 – RS–232								DDRB	56579
$DD04	Timer A								TAL	56580
$DD05									TAH	56581
$DD06	Timer B								TBL	56582
$DD07									TBH	56583
$DD0D				RS–232 IN			Timer Interrupt B	A	ICR	56589
$DD0E								Timer A Start	CRA	56590
$DD0F								Timer B Start	CRB	56591

* Connected but not used by O.S.

Commodore 64 Memory Map

Compiled by Jim Butterfield

Commodore 64 – RAM Memory Map

Hex	Decimal	Description	
0000	0	Chip directional register	
0001	1	Chip I/O, memory & tape control	
0003–0004	3–4	Float–Fixed vector	
0005–0006	5–6	Fixed–Float vector	
0007	7	Search character	
0008	8	Scan–quotes flag	
0009	9	TAB column save	
000A	10	0 = LOAD, 1 = VERIFY	
000B	11	Input buffer pointer/# subscrpt	
000C	12	Default DIM flag	
000D	13	Type: FF = string, 00 = numeric	
000E	14	Type: 80 = integer, 00 = floating point	
000F	15	DATA scan/LIST quote/memry flag	
0010	16	Subscript/FNx flag	
0011	17	0 = INPUT;$40 = GET;$98 = READ	
0012	18	ATN sign/Comparison eval flag	
0013	19	Current I/O prompt flag	
0014–0015	20–21	Integer value	
0016	22	Pointer: temporary string stack	
0017–0018	23–24	Last temp string vector	
0019–0021	25–33	Stack for temporary strings	
0022–0025	34–37	Utility pointer area	
0026–002A	38–42	Product area for multiplication	
002B–002C	43–44	Pointer: Start-of-Basic	
002D–002E	45–46	Pointer: Start-of-Variables	
002F–0030	47–48	Pointer: Start-of-Arrays	
0031–0032	49–50	Pointer: End-of-Arrays	
0033–0034	51–52	Pointer: String–storage(moving down)	
0035–0036	53–54	Utility string pointer	
0037–0038	55–56	Pointer: Limit–of–memory	
0039–003A	57–58	Current Basic line number	
003B–003C	59–60	Previous Basic line number	
003D–003E	61–62	Pointer: Basic statement for CONT	
003F–0040	63–64	Current DATA line number	
0041–0042	65–66	Current DATA address	
0043–0044	67–68	Input vector	
0045–0046	69–70	Current variable name	
0047–0048	71–72	Current variable address	
0049–004A	73–74	Variable pointer for FOR/NEXT	
004B–004C	75–76	Y–save; op–save; Basic pointer save	
004D	77	Comparison symbol accumulator	
004E–0053	78–83	Misc work area, pointers, etc	
0054–0056	84–86	Jump vector for functions	
0057–0060	87–96	Misc numeric work area	
0061	97	Accum#1: Exponent	
0062–0065	98–101	Accum#1: Mantissa	
0066	102	Accum#1: Sign	
0067	103	Series evaluation constant pointer	
0068	104	Accum#1 hi–order (overflow)	
0069–006E	105–110	Accum#2: Exponent, etc.	
006F	111	Sign comparison, Acc#1 vs #2	
0070	112	Accum#1 lo–order (rounding)	
0071–0072	113–114	Cassette buff len/Series pointer	
0073–008A	115–138	CHRGET subroutine; get Basic char	
007A–007B	122–123	Basic pointer (within subrtn)	
008B–008F	139–143	RND seed value	
0090	144	Status word ST	
0091	145	Keyswitch PIA: STOP and RVS flags	
0092	146	Timing constant for tape	
0093	147	Load = 0, Verify = 1	
0094	148	Serial output: deferred char flag	
0095	149	Serial deferred character	
0096	150	Tape EOT received	
0097	151	Register save	
0098	152	How many open files	
0099	153	Input device, normally 0	
009A	154	Output CMD device, normally 3	
009B	155	Tape character parity	
009C	156	Byte–received flag	
009D	157	Direct = $80/RUN = 0 output control	
009E	158	Tp Pass 1 error log/char buffer	
009F	159	Tp Pass 2 err log corrected	
00A0–00A2	160–162	Jiffy Clock HML	
00A3	163	Serial bit count/EOI flag	
00A4	164	Cycle count	
00A5	165	Countdown,tape write/bit count	
00A6	166	Tape buffer pointer	
00A7	167	Tp Wrt ldr count/Rd pass/inbit	
00A8	168	Tp Wrt new byte/Rd error/inbit cnt	
00A9	169	Wrt start bit/Rd bit err/stbit	
00AA	170	Tp Scan;Cnt;Ld;End/byte assy	
00AB	171	Wr lead length/Rd checksum/parity	
00AC–00AD	172–173	Pointer: tape bufr, scrolling	
00AE–00AF	174–175	Tape end adds/End of program	
00B0–00B1	176–177	Tape timing constants	
00B2–00B3	178–179	Pntr: start of tape buffer	
00B4	180	1 = Tp timer enabled; bit count	
00B5	181	Tp EOT/RS232 next bit to send	
00B6	182	Read character error/outbyte buf	
00B7	183	# characters in file name	
00B8	184	Current logical file	
00B9	185	Current secndy address	
00BA	186	Current device	
00BB–00BC	187–188	Pointer to file name	
00BD	189	Wr shift word/Rd input char	
00BE	190	# blocks remaining to Wr/Rd	
00BF	191	Serial word buffer	
00C0	192	Tape motor interlock	
00C1–00C2	193–194	I/O start address	
00C3–00C4	195–196	Kernel setup pointer	
00C5	197	Last key pressed	
00C6	198	# chars in keybd buffer	
00C7	199	Screen reverse flag	
00C8	200	End-of-line for input pointer	
00C9–00CA	201–202	Input cursor log (row, column)	
00CB	203	Which key: 64 if no key	
00CC	204	0 = flash cursor	
00CD	205	Cursor timing countdown	
00CE	206	Character under cursor	
00CF	207	Cursor in blink phase	
00D0	208	Input from screen/from keyboard	
00D1–00D2	209–210	Pointer to screen line	
00D3	211	Position of cursor on above line	
00D4	212	0 = direct cursor, else programmed	
00D5	213	Current screen line length	
00D6	214	Row where curosr lives	
00D7	215	Last inkey/checksum/buffer	
00D8	216	# of INSERTs outstanding	
00D9–00F2	217–242	Screen line link table	
00F3–00F4	243–244	Screen color pointer	
00F5–00F6	245–246	Keyboard pointer	
00F7–00F8	247–248	RS–232 Rcv pntr	
00F9–00FA	249–250	RS–232 Tx pntr	
00FF–010A	256–266	Floating to ASCII work area	
0100–013E	256–318	Tape error log	
0100–01FF	256–511	Processor stack area	
0200–0258	512–600	Basic input buffer	
0259–0262	601–610	Logical file table	
0263–026C	611–620	Device # table	
026D–0276	621–630	Sec Adds table	
0277–0280	631–640	Keybd buffer	
0281–0282	641–642	Start of Basic Memory	
0283–0284	643–644	Top of Basic Memory	
0285	645	Serial bus timeout flag	
0286	646	Current color code	
0287	647	Color under cursor	
0288	648	Screen memory page	
0289	649	Max size of keybd buffer	
028A	650	Repeat all keys	
028B	651	Repeat speed counter	
028C	652	Repeat delay counter	
028D	653	Keyboard Shift/Control flag	
028E	654	Last shift pattern	
028F–0290	655–656	Keyboard table setup pointer	
0291	657	Keyboard shift mode	
0292	658	0 = scroll enable	
0293	659	RS–232 control reg	
0294	660	RS–232 command reg	
0295–0296	661–662	Bit timing	
0297	663	RS–232 status	
0298	664	# bits to send	
0299–029A	665	RS–232 speed/code	
029B	667	RS232 receive pointer	
029C	668	RS232 input pointer	
029D	669	RS232 transmit pointer	
029E	670	RS232 output pointer	
029F–02A0	671–672	IRQ save during tape I/O	
02A1	673	CIA 2 (NMI) Interrupt Control	
02A2	674	CIA 1 Timer A control log	
02A3	675	CIA 1 Interrupt Log	
02A4	676	CIA 1 Timer A enabled flag	
02A5	677	Screen row marker	
02C0–02FE	704–766	(Sprite 11)	
0300–0301	768–769	Error message link	
0302–0303	770–771	Basic warm start link	
0304–0305	772–773	Crunch Basic tokens link	
0306–0307	774–775	Print tokens link	
0308–0309	776–777	Start new Basic code link	
030A–030B	778–779	Get arithmetic element link	
030C	780	SYS A–reg save	
030D	781	SYS X–reg save	
030E	782	SYS Y–reg save	
030F	783	SYS status reg save	
0310–0312	784–785	USR function jump	(B248)
0314–0315	788–789	Hardware interrupt vector	(EA31)
0316–0317	790–791	Break interrupt vector	(FE66)
0318–0319	792–793	NMI interrupt vector	(FE47)
031A–031B	794–795	OPEN vector	(F34A)
031C–031D	796–797	CLOSE vector	(F291)
031E–031F	798–799	Set–input vector	(F20E)
0320–0321	800–801	Set–output vector	(F250)
0322–0323	802–803	Restore I/O vector	(F333)
0324–0325	804–805	INPUT vector	(F157)
0326–0327	806–807	Output vector	(F1CA)
0328–0329	808–809	Test–STOP vector	(F6ED)
032A–032B	810–811	GET vector	(F13E)
032C–032D	812–813	Abort I/O vector	(F32F)
032E–032F	814–815	Warm start vector	(FE66)
0330–0331	816–817	LOAD link	(F4A5)
0332–0333	818–819	SAVE link	(F5ED)
033C–03FB	828–1019	Cassette buffer	
0340–037E	832–894	(Sprite 13)	
0380–03BE	896–958	(Sprite 14)	
03C0–03FE	960–1022	(Sprite 15)	
0400–07FF	1024–2047	Screen memory	
0800–9FFF	2048–40959	Basic RAM memory	
8000–9FFF	32768–40959	Alternate: ROM plug–in area	
A000–BFFF	40960–49151	ROM: Basic	
A000–BFFF	49060–59151	Alternate: RAM	
C000–CFFF	49152–53247	RAM memory, including alternate	
D000–D02E	53248–53294	Video Chip (6566)	
D400–D41C	54272–54300	Sound Chip (6581 SID)	
D800–DBFF	55296–56319	Color nybble memory	
DC00–DC0F	56320–56335	Interface chip 1, IRQ (6526 CIA)	
DD00–DD0F	56576–56591	Interface chip 2, NMI (6526 CIA)	
D000–DFFF	53248–53294	Alternate: Character set	
E000–FFFF	57344–65535	ROM: Operating System	
E000–FFFF	57344–65535	Alternate: RAM	
FF81–FFF5	65409–65525	Jump Table, Including:	
FFC6		– Set Input channel	
FFC9		– Set Output channel	
FFCC		– Restore default I/O channels	
FFCF		– INPUT	
FFD2		– PRINT	
FFE1		– Test Stop key	
FFE4		– GET	

Commodore 64 – ROM Memory Map

A000:	ROM control vectors
A00C:	Keyword action vectors
A052:	Function vectors
A080:	Operator vectors
A09E:	Keywords
A19E:	Error messages
A328:	Error message vectors
A365:	Misc messages
A38A:	Scan stack for FOR/GOSUB
A3B8:	Move memory
A3FB:	Check stack depth
A408:	Check memory space
A435:	'out of memory'
A437:	Error routine
A469:	BREAK entry
A474:	'ready.'
A480:	Ready for Basic
A49C:	Handle new line
A533:	Re–chain lines
A560:	Receive input line
A579:	Crunch tokens
A613:	Find Basic line
A642:	Perform [NEW]
A65E:	Perform [CLR]
A68E:	Back up text pointer
A69C:	Perform [LIST]
A742:	Perform [FOR]
A7ED:	Execute statement
A81D:	Perform [RESTORE]
A82C:	Break
A82F:	Perform [STOP]
A831:	Perform [END]
A857:	Perform [CONT]
A871:	Perform [RUN]
A883:	Perform [GOSUB]
A8A0:	Perform [GOTO]
A8D2:	Perform [RETURN]
A8F8:	Perform [DATA]
A906:	Scan for next statement
A928:	Perform [IF]
A93B:	Perform [REM]
A94B:	Perform [ON]
A96B:	Get fixed point number
A9A5:	Perform [LET]
AA80:	Perform [PRINT#]
AA86:	Perform [CMD]
AAA0:	Perform [PRINT]
AB1E:	Print string from (y.a)
AB3B:	Print format character
AB4D:	Bad input routine
AB7B:	Perform [GET]
ABA5:	Perform [INPUT#]
ABBF:	Perform [INPUT]
ABF9:	Prompt & input
AC06:	Perform [READ]
ACFC:	Input error messages
AD1E:	Perform [NEXT]
AD78:	Type match check
AD9E:	Evaluate expression
AEA8:	Constant – pi
AEF1:	Evaluate within brackets
AEF7:	')'
AEFF:	comma..
AF08:	Syntax error
AF14:	Check range
AF28:	Search for variable
AFA7:	Setup FN reference
AFE6:	Perform [OR]
AFE9:	Perform [AND]
B016:	Compare
B081:	Perform [DIM]
B08B:	Locate variable
B113:	Check alphabetic
B11D:	Create variable
B194:	Array pointer subrtine
B1A5:	Value 32768
B1B2:	Float–fixed
B1D1:	Set up array
B245:	'bad subscript'
B248:	'illegal quantity'
B34C:	Compute array size
B37D:	Perform [FRE]
B391:	Fix–float
B39E:	Perform [POS]
B3A6:	Check direct
B3B3:	Perform [DEF]
B3E1:	Check fn syntax
B3F4:	Perform [FN]
B465:	Perform [STR$]
B475:	Calculate string vector
B487:	Set up string
B4F4:	Make room for string
B526:	Garbage collection
B5BD:	Check salvageability
B606:	Collect string
B63D:	Concatenate
B67A:	Build string to memory
B6A3:	Discard unwanted string
B6DB:	Clean descriptor stack
B6EC:	Perform [CHR$]
B700:	Perform [LEFT$]
B72C:	Perform [RIGHT$]
B737:	Perform [MID$]
B761:	Pull string parameters
B77C:	Perform [LEN]
B782:	Exit string–mode
B78B:	Perform [ASC]
B79B:	Input byte paramter
B7AD:	Perform [VAL]
B7EB:	Parameters for POKE/WAIT
B7F7:	Float–fixed
B80D:	Perform [PEEK]
B824:	Perform [POKE]
B82D:	Perform [WAIT]
B849:	Add 0.5
B850:	Subtract–from
B853:	Perform [subtract]
B86A:	Perform [add]
B947:	Complement FAC#1
B97E:	'overflow'
B983:	Multiply by zero byte
B9EA:	Perform [LOG]
BA2B:	Perform [multiply]
BA59:	Multiply–a–bit
BA8C:	Memory to FAC#2
BAB7:	Adjust FAC#1/#2
BAD4:	Underflow/overflow
BAE2:	Multiply by 10
BAF9:	+ 10 in floating pt
BAFE:	Divide by 10
BB12:	Perform [divide]
BBA2:	Memory to FAC#1
BBC7:	FAC#1 to memory
BBFC:	FAC#2 to FAC#1
BC0C:	FAC#1 to FAC#2
BC1B:	Round FAC#1
BC2B:	Get sign
BC39:	Perform [SGN]
BC58:	Perform [ABS]
BC5B:	Compare FAC#1 to mem
BC9B:	Float–fixed
BCCC:	Perform [int]
BCF3:	String to FAC
BD7E:	Get ascii digit
BDC2:	Print 'IN..'
BDCD:	Print line number
BDDD:	Float to ascii
BF16:	Decimal constants
BF3A:	TI constants
BF71:	Perform [SQR]
BF7B:	Perform [power]
BFB4:	Perform [negative]
BFED:	Perform [EXP]
E043:	Series eval 1
E059:	Series eval 2
E097:	Perform [RND]
E0f9:	?? breakpoints ??
E12A:	Perform [SYS]
E156:	Perform [SAVE]
E165:	Perform [VERIFY]
E168:	Perform [LOAD]
E1BE:	Perform [OPEN]
E1C7:	Perform [CLOSE]
E1D4:	Parameters for LOAD/SAVE
E206:	Check default parameters
E20E:	Check for comma
E219:	Parameters for open/close
E264:	Perform [COS]
E26B:	Perform [SIN]
F2b4:	Perform [TAN]
E30E:	Perform [ATN]
E37B:	Warm restart
E394:	Initialize
E3A2:	CHRGET for zero page
E3BF:	Initialize Basic
E447:	Vectors for $300
E453:	Initialize vectors
E45F:	Power–up message
E500:	Get I/O address
E505:	Get screen size
E50A:	Put/get row/column
E518:	InitializeI/O
E544:	Clear screen
E566:	Home cursor
E56C:	Set screen pointers
E5A0:	Set I/O defaults
E5B4:	Input from keyboard
E632:	Input from screen
E684:	Quote test
E691:	Setup screen print
E6B6:	Advance cursor
E6ED:	Retreat cursor
E701:	Back into previous line
E716:	Output to screen
E87C:	Go to next line
E891:	Perform <return>
E8A1:	Check line decrement
E8B3:	Check line increment
E8CB:	Set color code
E8DA:	Color code table
E8EA:	Scroll screen
E965:	Open space on screen
E9C8:	Move a screen line
E9E0:	Synchronize color transfer
E9F0:	Set start–of–line
E9FF:	Clear screen line
EA13:	Print to screen
EA24:	Synchronize color pointer
EA31:	Interrupt – clock etc
EA87:	Read keyboard
EB79:	Keyboard select vectors
EB81:	Keyboard 1 – unshifted
EBC2:	Keyboard 2 – shifted
EC03:	Keyboard 3 – 'comm'
EC44:	Graphics/text contrl
EC4F:	Set graphics/text mode
EC78:	Keyboard 4
ECB9:	Video chip setup
ECE7:	Shift/run equivalent
ECF0:	Screen In address low
ED09:	Send 'talk'
ED0C:	Send 'listen'
ED40:	Send to serial bus
EDB2:	Serial timeout
EDB9:	Send listen SA
EDBE:	Clear ATN
EDC7:	Send talk SA
EDCC:	Wait for clock
EDDD:	Send serial deferred
EDEF:	Send 'untalk'
EDFE:	Send 'unlisten'
EE13:	Receive from serial bus
EE85:	Serial clock on
EE8E:	Serial clock off
EE97:	Serial output '1'
EEA0:	Serial output '0'
EEA9:	Get serial in & clock
EEB3:	Delay 1 ms
EEBB:	RS–232 send
EF06:	Send new RS–232 byte
EF2E:	No–DSR error
EF31:	No–CTS error
EF3B:	Disable timer
EF4A:	Compute bit count
EF59:	RS232 receive
EF7E:	Setup to receive
EFC5:	Receive parity error
EFCA:	Receive overflow
EFCD:	Receive break
EFD0:	Framing error
EFE1:	Submit to RS232
F00D:	No–DSR error
F017:	Send to RS232 buffer
F04D:	Input from RS232
F086:	Get from RS232
F0A4:	Check serial bus idle
F0BD:	Messages
F12B:	Print if direct
F13E:	Get..
F14E:	..from RS232
F157:	Input
F199:	Get.. tape/serial/rs232
F1CA:	Output..
F1DD:	..to tape
F20E:	Set input device
F250:	Set output device
F291:	Close file
F30F:	Find file
F31F:	Set file values
F32F:	Abort all files
F333:	Restore default I/O
F34A:	Do file open
F3D5:	Send SA
F409:	Open RS232
F49E:	Load program
F5AF:	'searching'
F5C1:	Print filename
F5D2:	'loading/verifying'
F5DD:	Save program
F68F:	Print 'saving'
F69B:	Bump clock
F6BC:	Log PIA key reading
F6DD:	Get time
F6E4:	Set time
F6ED:	Check stop key
F6FB:	Output error messages
F72D:	Find any tape headr
F76A:	Write tape header
F7D0:	Get buffer address
F7D7:	Set buffer start/end pointers
F7EA:	Find specific header
F80D:	Bump tape pointer
F817:	'press play..'
F82E:	Check tape status
F838:	'press record..'
F841:	Initiate tape read
F864:	Initiate tape write
F875:	Common tape code
F8D0:	Check tape stop
F8E2:	Set read timing
F92C:	Read tape bits
FA60:	Store tape chars
FB8E:	Reset pointer
FB97:	New character setup
FBA6:	Send transition to tape
FBC8:	Write data to tape
FBCD:	IRQ entry point
FC57:	Write tape leader
FC93:	Restore normal IRQ
FCB8:	Set IRQ vector
FCCA:	Kill tape motor
FCD1:	Check r/w pointer
FCDB:	Bump r/w pointer
FCE2:	Power reset entry
FD02:	Check 8–rom
FD10:	8–rom mask
FD15:	Kernal reset
FD1A:	Kernal move
FD30:	Vectors
FD50:	Initialize system constnts
FD9B:	IRQ vectors
FDA3:	Initialize I/O
FDDD:	Enable timer
FDF9:	Save filename data
FE00:	Save file details
FE07:	Get status
FE18:	Flag status
FE1C:	Set status
FE21:	Set timeout
FE25:	Read/set top of memory
FE27:	Read top of memory
FE2D:	Set top of memory
FE34:	Read/set bottom of memory
FE43:	NMI entry
FE66:	Warm start
FEB6:	Reset IRQ & exit
FEBC:	Interrupt exit
FEC2:	RS–232 timing table
FED6:	NMI RS–232 in
FF07:	NMI RS–232 out
FF43:	Fake IRQ
FF48:	IRQ entry
FF81:	Jumbo jump table
FFFA:	Hardware vectors

CIA 1 (IRQ) (6526)

$DC00	Paddle Sel A	B		Joystick 0 Fire	Right	Left	Down	Up	PRA	56320
	Keyboard Row Select (inverted)									
$DC01				Joystick 1 Fire	Right	Left	Down	Up	PRB	56321
	Keyboard Column Read									
$DC02	$FF – All Output								DDRA	56322
$DC03	$00 – All Input								DDRB	56323
$DC04	Timer A								TAL	56324
$DC05									TAH	56325
$DC06	Timer B								TBL	56326
$DC07									TBH	56327
~										
$DC0D				Tape Input			Timer Interrupt B	A	ICR	56333
$DC0E					One Shot	Out Mode	Time PB6 Out	Timer A Start	CRA	56334
$DC0F					One Shot	Out Mode	Time PB7 Out	Timer B Start	CRB	56335

CIA 2 (NMI) (6526)

$DD00	Serial IN	Clock IN	Serial OUT	Clock OUT	ATN OUT	RS-232 OUT	$\overline{\text{VIC II}}$ addr 15	$\overline{\text{VIC II}}$ addr 14	PRA	56576
$DD01	DSR IN	CTS IN		DCD* IN	RI* IN	DTR OUT	RTS OUT	RS-232 IN	PRB	56577
$DD02	$3F – Serial								DDRA	56578
$DD03	$00 – P.U.P. All Input or $06 – RS-232								DDRB	56579
$DD04	Timer A								TAL	56580
$DD05									TAH	56581
$DD06	Timer B								TBL	56582
$DD07									TBH	56583
~										
$DD0D				RS-232 IN			Timer Interrupt B	A	ICR	56589
$DD0E								Timer A Start	CRA	56590
$DD0F								Timer B Start	CRB	56591

* Connected but not used by O.S.

Processor I/O Port (6510)

$0000	IN	IN	OUT	IN	OUT	OUT	OUT	OUT	DDR 0
$0001			Tape Motor	Tape Sense	Tape Write	D-ROM Switch	EF RAM Switch	AB RAM Switch	PR 1

SID (6581)

Voice 1	Voice 2	Voice 3										Voice 1	Voice 2	Voice 3
$D400	$D407	$D40E	Frequency								L	54272	54279	54286
$D401	$D408	$D40F									H	54273	54280	54287
$D402	$D409	$D410	Pulse Width								L	54274	54281	54288
$D403	$D40A	$D411	0	0	0	0					H	54275	54282	54289
$D404	$D40B	$D412	Voice Type NSE	PUL	SAW	TRI				Key		54276	54283	54290
$D405	$D40C	$D413	Attack Time 2ms – 8ms				Decay Time 6ms – 24 sec					54277	54284	54291
$D406	$D40D	$D414	Sustain Level				Release Time 6ms 24 sec					54278	54285	54292

Voices (write only)

$D415	0	0	0	0	0			L	54293
$D416	Filter Frequency							H	54294
$D417	Resonance				Filter Voices Ext	V3	V2	V1	54295
$D418	V3 off	Passband HI	BP	LO	Master Volume				54296

Filter & Volume (write only)

$D419	Paddle X (A/D #1)	54297
$D41A	Paddle Y (A/D #2)	54298
$D41B	Noise 3 (random)	54299
$D41C	Envelope 3	54300

Sense (read only)

Note: Special Voice Features
(TEST, RING MOD, SYNC)
are omitted from the above diagram.

Page 22 was incorrectly printed in issue 7. This page is correct.

type, they are performed in order from left to right. We can use parentheses to show you better how the VIC actually sets up this calculation. The VIC performs the calculations in order as shown and displays 11 as the answer.

Calculation Trail	Specific Operation Each Time
10*(2↑2)/4+3−2	2*2=4
(10*4)/4+3−2	10*4=40
(40/4)+3−2	40/4=10
(10+3)−2	10+3=13
(13−2)	13−2=11

You could **change** this calculation by adding parentheses, like this:

```
PRINT10*2↑2/(4+3−2)
```

In this case, the VIC will first perform the exponentiation, then the multiplication . . . but before dividing, it will do the parenthetical operation. The calculation trail now looks like this:

10*2↑2/(4+3−2)	(4+3−2)= 5
10*2↑2/5	2↑2= 4
(10*4)/5	10*4=40
40/5	40/5= 8

Using Numbers in Programs

This part of our "magical" VIC tour is going to show you how to use numbers in your programs.

You can use numbers in your BASIC programs—not only directly, like you'd use numbers in counting or calculating—but also **indirectly** in the form of **variables** that can change as a number changes.

Our previous section gave you a quick introduction to calculation on the VIC. We mostly worked in the DIRECT MODE. Our next step is to explore how to use numbers in your BASIC PROGRAMS.

Let's begin with a short example.

```
10 INPUTX        (and hit RETURN)
20 PRINTX*10     (and hit RETURN)
30 GOTO10        (and hit RETURN)
```

Type RUN to start the program. Now type **any number** and hit RETURN. The VIC multiples your number by 10 and gives you the result. If you wanted to multiply your number by 10 percent, or by pi or by any other number, all you have to do is change the 10 in line 20 to something else. Also, the calculation in line 20 could also be division, addition, subtraction, or any other calculation.

The key to this program is the INPUT statement in line 10, which accepts the number *YOU* type in, and assigns it the value X. After it multiplies your number times 10 and PRINTs the result, the program loops back and asks for another number, which becomes the "new X."

To really understand how this program works . . . and in fact how most number-oriented programs work . . . you should understand **numeric variables.**

Numeric Variables

The concept of variables is explained in the VIC user manual, but this is one of the hardest computing concepts to grasp, so we're going to talk you through it gradually.

A variable is like a code. Numeric variables are like codes the VIC uses— and which you can use—to stand for a number. Numeric variables help the VIC remember and manipulate numbers—even change them—in a program. You can use variables for words, letters, phrases and graphics, too, but in this discussion we'll concentrate on those variables we use to represent numbers.

There are LEGAL and ILLEGAL variable names. Numeric variable names can be a single letter, two letters, or a letter and a number. Examples are: A, X, AA, AB, P1, R4, AB2, MU5. A special kind of numeric variable, which limits the value to an integer (whole number, no fractions or decimals) has a % sign as the last character (Examples: A%, AB%, P1%, R4%).

The other type of variable is called a "string variable." Here, numbers are used like words instead of **values** (e.g., a zip code or phone number). String variables end in a $ sign (Examples: A$, AB$) and are mostly used to specify letters, words, phrases, graphic symbols and descriptive numbers. If you tell the VIC that X=19406 then the VIC interprets the X as the **value** 19,406. But if you want that number to be descriptive, like a zip code, then you tell the VIC that X$=19406 and the number 19406 will be used like a word or phrase instead of a value.

How Variables Work

If I say X=10 then I have just created a **variable** and from now on the letter X stands for the number 10. You might ask why we don't just use the number 10 but in a moment we'll show you why using an X gives the VIC more calculating power and program flexibility.

If I tell the VIC X=10, then any time I use the variable X in my program, the VIC will think of it as the number 10. If I tell the VIC to PRINTX, the VIC PRINTs the number 10. Let's test this assumption. Type this:

```
NEW          (and hit RETURN—to erase
               previous program)
X=10         (and hit RETURN)
```

We've told the VIC that the variable X equals 10. We can do this in DIRECT (IMMEDIATE) mode, without using line numbers. The VIC will automatically store these variables in its memory . . . although the variables may be changed or erased if we RUN a program. Let's continue. Type this:

```
PRINTX       (and hit RETURN)
```

The VIC responds by displaying the number 10. That's because we created the X variable

MAKE THE MOST OF YOUR COMMODORE

HIGH-RESOLUTION GRAPHICS

Now you can give your PET/CBM a High-Resolution Graphics capability with the MTU Graphics Hardware and Software Package. The Hardware is easily installed and the new graphics board provides five EXTRA ROM Sockets and 8k RAM MEMORY EXPANSION which can be used for program or data storage when graphics are not required. A powerful graphics Software Package is included and contains many extra BASIC commands for drawing lines defining shapes etc. The Graphics Hardware does not affect normal operation of the Commodore.

Available on all PET/CBM - BASIC 1, 2, 3, or 4.

RS232 TEST SET

The RS232 TESTSET is a small hand held device that connects inline with the interface cable, the terminal or the modem and monitors the line signals. The TESTSET passes all 25 lines through and so can be left connected without affecting communications. It is completely portable as no batteries are required since power us derived from the interface signals. Each indicator circuit is current limited to meet the requirements of the RS232 Interface Standard. Also the voltage range for activating the bright LED display corresponds with this standard and thereby reduces troubleshooting to a "GO-NOGO" problem instead of trying to measure active signals to determine voltage levels. One 2-pin and one 3-pin jumper are included.

PRINTOUT

Don't forget your PRINTOUT Magazine Subscription - for Commodore PET/CBM Lovers 12 issues p.a. from Jan. '82 incl. postage - $50.00 p.a.

VIC COMPUTING

For all VIC 20 owners, this sister to PRINTOUT is a must. 6 issues p.a. incl. postage from Jan. '82 - $25.00 p.a.

PROGRAMMERS TOOLKIT ROMS

These ROMs plug into spare sockets in your PET/CBMs and give the user additional commands such as TRACE, single STEP, FIND, RE-NUMBER, AUTO line numbering, DUMP variable contents, APPEND, and DELETE multiple lines. Also available are the DISK-O-PRO Tool-kits which gives all the extra DOS 2.0 commands to DOS 1.0 users as well as commands like PRINT USING, SCROLL and disk program MERGE -25 commands in all. The COMMAND-O provides DISK-O-PRO commands for BASIC 4.0 users.

The Programmers Toolkit for BASIC 1.0/2.0/3.0/4.0 users.
DISK-O-PRO Toolkit for BASIC 2.0/3.0 users.
COMMAND-O Toolkit for BASIC 4.0 users.

Tel: (03) 614 1433
614-1551
Telex: AA 30333.

MICROCOMP
MICROCOMPUTER SYSTEMS DESIGNERS

B.S. MICROCOMP
PTY. LIMITED,
4th & 3rd Floors,
561 Bourke Street,
Melbourne, 3000.

....NEWSF

SFLASH....NEWSFLAS

ASH....NEWSFLASH....NE

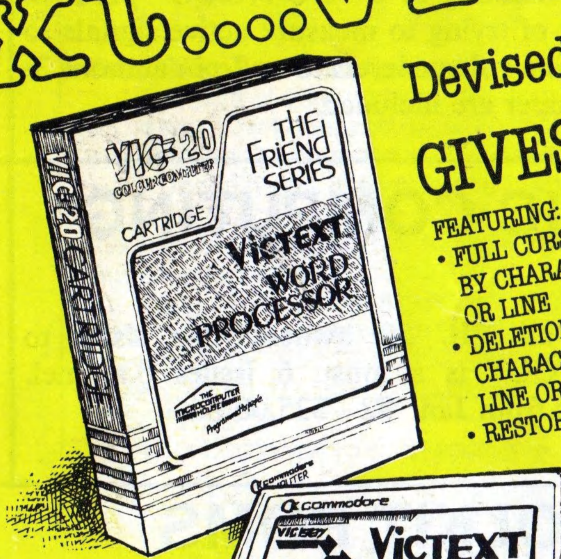
ATTENTION VIC 20 OWNERS

Inexpensive text editor word processor cartridge now available.

ext....Victext....Victe

Devised by The Microcomputer House.

GIVES 8K OF ADDITIONAL RAM


VIC-20
COLOUR COMPUTER
THE FRIEND SERIES
CARTRIDGE
VICTEXT WORD PROCESSOR
VIC 20 CARTRIDGE
THE MICROCOMPUTER HOUSE
Programmed for people
commodore
VICTEXT

FEATURING:

- FULL CURSOR CONTROL BY CHARACTER, WORD OR LINE
- DELETION BY CHARACTER, WORD, LINE OR BLOCK OF TEXT
- RESTORE DELETED TEXT
- DUPLICATE A BLOCK OF TEXT
- TABULATIONS
- RIGHT JUSTIFICATION WHEN PRINTING
- PROGRAMMING THE TEXT EDITOR
- SEARCH FOR A WORD AND REPLACE WITH ANOTHER
- SAVE AND RETRIEVE TEXT FROM CASSETTE OR DISC
- MERGE TEXT WITH ANOTHER FROM CASSETTE OR DISC
- PROGRAMMED IN MACHINE CODE FOR MAXIMUM SPEED AND EFFICIENCY

Flexible, economic, effective. Ideal for busines[illegible]men, writers, students, enthusiasts.

INCREDIBLE VALUE AT ONLY $14[illegible]00

THE
MICROCOMPUTER
HOUSE

Programmed for people.

The Microcomputer House is at 116 Abercrombie Street, Chippendale. Telephone 698 7076

P.S. Want to learn to play the guitar? We've got a programme for only $19.95 that can teach you.